universität wien

DISSERTATION

Titel der Dissertation

Efficient Feature Reduction and Classification Methods

Applications in Drug Discovery and Email Categorization

Verfasser

Mag. Andreas Janecek

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, im Dezember 2009

Studienkennzahl lt. Studienblatt:	A 786 880
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Priv.-Doz. Dr. Wilfried Gansterer

Contents

Summary

The sheer volume of data today and its expected growth over the next years are some of the key challenges in data mining and knowledge discovery applications. Besides the huge number of data samples that are collected and processed, the high dimensional nature of data arising in many applications causes the need to develop effective and efficient techniques that are able to deal with this massive amount of data. In addition to the significant increase in the demand of computational resources, those large datasets might also influence the quality of several data mining applications (especially if the number of features is very high compared to the number of samples). As the dimensionality of data increases, many types of data analysis and classification problems become significantly harder. This can lead to problems for both supervised and unsupervised learning. Dimensionality reduction and feature (subset) selection methods are two types of techniques for reducing the attribute space. While in feature selection a subset of the original attributes is extracted, dimensionality reduction in general produces linear combinations of the original attribute set. In both approaches, the goal is to select a low dimensional subset of the attribute space that covers most of the information of the original data. During the last years, feature selection and dimensionality reduction techniques have become a real prerequisite for data mining applications.

There are several open questions in this research field, and due to the often increasing number of candidate features for various application areas (e. g., email filtering or drug classification/molecular modeling) new questions arise. In this thesis, we focus on some open research questions in this context, such as the relationship between feature reduction techniques and the resulting classification accuracy and the relationship between the variability captured in the linear combinations of dimensionality reduction techniques (e. g., PCA, SVD) and the accuracy of machine learning algorithms operating on them. Another important goal is to better understand new techniques for dimensionality reduction, such as nonnegative matrix factorization (NMF), which can be applied for finding parts-based, linear representations of nonnegative data. This "sum-of-parts" representation is especially useful if the interpretability of the original data should be retained. Moreover, performance aspects of feature reduction algorithms are investigated. As data grow, implementations of feature selection and dimensionality reduction techniques for high-performance parallel and distributed computing environments become more and more important.

In this thesis, we focus on two types of open research questions: methodological advances without any specific application context, and application-driven advances for a specific application context. Summarizing, new methodological contributions are the following: The utilization of nonnegative matrix factorization in the context of classification methods is investigated. In particular, it is of interest how the improved interpretability of NMF factors due to the non-negativity constraints (which is of central importance in various problem settings) can be exploited. Motivated by this problem context two new fast initialization techniques for NMF based on feature selection are introduced. It is shown how approximation accuracy can be increased and/or how computational effort can be reduced compared to standard randomized seeding of the NMF and to state-of-the-art initialization strategies suggested earlier. For example, for a given number of iterations and a required approximation error a speedup of 3.6 compared to standard initialization, and a speedup of 3.4 compared to state-of-the-art initialization strategies could be achieved. Beyond that, novel classification methods based on the NMF are proposed and investigated. We can show that they are not only competitive in terms of classification accuracy with state-of-the-art classifiers, but also provide important advantages in terms of computational effort (especially for low-rank approximations). Moreover, parallelization and distributed execution of NMF is investigated. Several algorithmic variants for efficiently computing NMF on multi-core systems are studied and compared to each other. In particular, several approaches for exploiting task and/or data-parallelism in NMF are studied. We show that for some scenarios new algorithmic variants clearly outperform existing implementations. Last, but not least, a computationally very efficient adaptation of the implementation of the ALS algorithm in Matlab 2009a is investigated. This variant reduces the runtime significantly (in some settings by a factor of 8) and also provides several possibilities to be executed concurrently.

In addition to purely methodological questions, we also address questions arising in the adaptation of feature selection and classification methods to two specific application problems: email classification and in silico screening for drug discovery. Different research challenges arise in the contexts of these different application areas, such as the dynamic nature of data for email classification problems, or the imbalance in the number of available samples of different classes for drug discovery problems. Application-driven advances of this thesis comprise the adaptation and application of latent semantic indexing (LSI) to the task of email filtering. Experimental results show that LSI achieves significantly better classification results than the widespread de-facto standard method for this special application context. In the context of drug discovery problems, several groups of well discriminating descriptors could be identified by utilizing the "sum-of-parts" representation of NMF. The number of important descriptors could be further increased when applying sparseness constraints on the NMF factors.

Chapter 1

Introduction

The tremendous improvements in techniques for collecting, storing and transferring large volumes of data have also increased the volume of data for knowledge discovery and data mining applications. Data grow not only due to the number of data samples available, but also due to the increasing number of candidate features for various application areas. Not only the effort and computational cost of data mining applications grow with increasing dimension of data.

1.1 Motivation and Problem Statement

As the dimensionality of the feature space increases, many types of data analysis and classification also become significantly harder, and, additionally, the data becomes increasingly sparse in the space it occupies which can lead to big difficulties for both supervised and unsupervised learning. This phenomenon – known as the *curse of dimensionality* – is based on the fact that high dimensional data is often difficult to work with. A large number of features can increase the noise of the data and thus the error of a learning algorithm, especially if there are only few observations (i. e., data samples) compared to the number of features.

Feature selection and dimensionality reduction methods (summarized as *feature reduction* methods) are two techniques that aim at solving these problems by reducing the number of features and thus the dimensionality of the data. In the last years, several studies have focused on improving feature selection and dimensionality reduction techniques and substantial progress has been obtained in selecting, extracting and constructing useful feature sets. However, due to the strong influence of different feature reduction methods on the classification accuracy, there are still several open questions in this research field. Moreover, due to the often increasing number of candidate features for various application areas new questions arise.

In this thesis, we focus on some of these open research questions, such as the relationship between several feature reduction techniques and the resulting classification accuracy. The goal is to identify a set of features that best approximate the original data without a reduction in the classification result. Other problems are based on computational cost of feature reduction algorithms. The huge amount of data arises the need to develop computationally efficient feature reduction techniques which can be performed concurrently. To address this problem we study several approaches for exploiting task and/or data-parallelism in NMF and introduce computationally very efficient adaptations of the existing NMF algorithms. To further speed up the runtime of NMF, we investigate new initialization techniques for NMF based on feature selection as well as fast and effective classification methods based on NMF. Moreover, there are several problems when analyzing the interpretability of dimensionality reduction techniques. The linear combination of dimensionality reduction methods are usually not interpretable and the information how much an original feature contributes is often lost. In this thesis, we investigate how the improved interpretability of NMF factors due to the non-negativity constraints can be exploited to retain interpretability of the original data.

Experimental evaluations are performed on datasets coming from two very different application areas with different research challenges: email classification and in silico screening for drug discovery.

1.2 Synopsis

This work is structured into two parts, each several chapters.

In Part I theoretical background is summarized and discussed. In Chapter 2 the definitions of data mining and knowledge discovery are discussed, connections of data mining to other disciplines are summarized, and a short overview about the different types of data used in data mining applications is provided. Moreover, a new knowledge discovery process model (KDP) is introduced and the five steps of this model together with several feedback loops are discussed in detail. Finally, there is a summary of relevant introductory literature for data mining and knowledge discovery. Feature reduction methods – which are included in step 3 of the KDP model introduced in Chapter 2 – are discussed in detail in *Chapter 3*. Definition for both feature selection and dimensionality reduction techniques are given and several important methods are analyzed and compared. In *Chapter 4* several supervised learning algorithms are reviewed and compared in terms of expected classification accuracy, comprehensibility, complexity and speed. In the KDP model introduced in Chapter 2, supervised learning algorithms are included in step 4 together with unsupervised learning methods such as clustering algorithms. *Chapter 5* discusses the two application areas that are used in Part II of this thesis. These two application areas are on the one hand email filtering, where the feature space contains various properties of email messages, and the other hand, drug discovery problems are considered. Here, quantitative representations of molecular structures are encoded in terms of information-preserving descriptor values. Relevant literature for both application areas is summarized and the classification problems for the two fields are discussed. Finally, general characteristics of these areas are compared.

In Part II of this thesis, we focus on three dimensionality reduction methods discussed in Chapter 3 (PCA, SVD, NMF) for investigating the interaction between feature reduction methods and learning algorithms in the context of the two application areas email filtering and drug discovery. In *Chapter 6* the relationship between several feature reduction methods and the resulting classification accuracy is investigated. Classification results based on different variants of the principal component analysis, information gain, and deterministic wrapper methods are compared for the two application areas discussed in Chapter 5. In *Chapter 7* the application of latent semantic indexing to the task of spam filtering is studied. Comparisons to the basic vector space model and to a state-of-the-art rule-based filtering system are given for two different feature sets. In *Chapter 8* new initialization strategies for nonnegative matrix factorization algorithms are introduced. Besides than, the interpretability of the NMF factors in the email classification and drug discovery contexts are also considered. Motivated by the approaches investigated in Chapter 8, two new classification methods based on the nonnegative matrix factorization are introduced and investigated in *Chapter 9*. Finally, *Chapter 10* introduces several ideas to execute NMF computations task- and data-parallel. Complex interaction between problem size, initialization strategy, accuracy of the factorization and computational efficiency of NMF computations on multi-core systems are illustrated. Moreover, a computationally very efficient adaption of a specific NMF algorithm is introduced.

1.3 Summary of Publications

The study described in Chapter 6 was presented at the 2008 Workshop on New Challenges for Feature Selection in Data Mining and Knowledge Discovery in conjunction with the ECML PKDD Conference 2008. This work was further published in the Journal of Machine Learning Research (JMLR) series for Workshop and Conference Proceedings, Volume 4 (cf. Reference [JGD08]). The study on the application of LSI to the task of spam filtering (Chapter 7) was presented at the 2007 Text

Mining Workshop in conjunction with the SIAM SDM 2007. Moreover, this work was published as a separate chapter in the book Survey of Text Mining II (see Reference [GJN08]). New initialization strategies for NMF (cf. Chapter 8) and new approaches of utilizing NMF for e-mail classification problems (cf. Chapter 9) were presented at the 2009 Text Mining Workshop at the SIAM SDM 2009 (see Reference[JG09]). This work will further appear as a separate chapter in the book Survey of Text Mining III.

1.4 Notation

In this thesis, we use the following notation: A matrix is represented as an uppercase italic letter (example: A, B, Σ, $\dots$, null matrix O, identity matrix I). A vector is represented as a lowercase bold letter (example: $\mathbf{u}$, $\mathbf{x}$, $\mathbf{q}_1$, $\dots$, null vector $\mathbf{o}$). A scalar is represented as a lowercase Greek letter (example: λ, μ, $\dots$), and indices (positive integers) as lowercase italic letters i, j, n, m, and p.

1.5 Acknowledgements

First and foremost, I want to express my special gratitude to my supervisor Dr. Wilfried Gansterer, for the support, encouragement and guidance he showed me throughout the last years. Wilfried also broadened my perspective in several aspects by sharing his theoretical, methodological, and empirical knowledge with me. He has not only given me constructive criticism which I hope I have been able to apply in my work, but he has also been a great source of inspiration to me.

Moreover, I want to thank Prof. Gerhard Ecker and his group at the Department of Medicinal Chemistry for the productive cooperation and joint work during the last years. Many thanks also to my colleagues at my department for their encouragement and practical advice. I am thankful to everyone who helped out reading my reports, commenting on my views and helping me understand and enrich my ideas.

Many friends have helped me stay sane through the last weeks of writing this thesis. I greatly value their friendship and I appreciate their belief in me. A special thanks to all friends who helped me proofreading important parts of this thesis.

None of this would have been possible without the love and support of my family. I want to express my heartfelt gratitude to my parents who always supported my interest in science and research, and always supported decisions I made in my life. Finishing this work also reminds me of my grand-fathers a lot, who would have loved seeing their grandson finishing his PhD thesis.

Part I

Theoretical Background

Chapter 2

Data Mining and Knowledge Discovery

Data mining (DM) is a general term that combines various computer-based procedures to analyze possibly huge datasets. It is an interdisciplinary research area where experts from various areas such as statistics, computer science, mathematics etc. often work together with experts from different application areas. DM techniques are usually deployed to find novel and useful patterns in data that might otherwise remain unknown. A frequently cited definition is given by Decker and Focardi [DF95, p. 3]: *Data mining is a problem-solving methodology that finds a logical or mathematical description, eventually of a complex nature, of patterns and regularities in a set of data.* Contrary to traditional statistical methods that are used to verify or disprove a pre-defined hypothesis, data mining offers the possibility to automatically generate new hypotheses. The goal is to efficiently and effectively extract information and knowledge from data that should *make sense* of the data, i. e., this knowledge should exhibit some essential attributes: it should be understandable, valid, novel and useful (cf. [CPS07]).

By definition, data mining is a part of a superordinated process called **knowledge discovery in databases (KDD)**, where the term "database" refers to any kind of data storage and does not solely comprise data stored in database management systems. KDD is defined by Fayyad *et al.* [FPS96, p. 6] as follows: *Knowledge discovery in databases describes the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.* In other words, KDD addresses the problem of mapping low-level data (which are typically too large to understand) into other forms that might be more compact, more abstract (e. g., a descriptive approximation of the data) or more useful (e. g., a predictive model for estimating the values of future cases). In the definition of Fayyad *et al.* [FPS96], the application of specific DM methods is at the core of the knowledge discovery process (KDP), which also comprises the extraction of background knowledge of the respective application area, the problem/goal definition of the whole KDP, the selection and preparation of the data, the reduction of features and samples, the selection of the learning model and interpretation and evaluation of the obtained results. Data mining is integrated as a single step of the KDP, usually between model selection and interpretation. In the literature, the term *data mining* is often used instead of or synonymously with the term *knowledge discovery in databases* and thus sometimes refers to the complete KDP. In this thesis, we follow this common terminology and use the terms *data mining* and *data mining application* to refer to the complete KDP. Contrary to that, the terms *data mining algorithm* and *data mining method* are used to refer to the actual data mining step within the KDP.

In this chapter, all relevant literature is summarized in a separate section at the end. However, when it seems useful the references are also given directly in the text as well.

Data mining applications. Several DM applications are used in a wide variety of scientific and commercial applications. For example, web or media search algorithms are often based on DM techniques, such as text mining or document clustering, (mass) surveillance, pattern mining and pattern recognition. Pharmaco- and bioinformatics use DM for diagnosis, QSAR-modeling or gene expression, while other scientific fields use it for image screening or spatial DM (application of DM techniques to spatial data). In costumer relationship management applications, automated DM applications are often used for decision support, market basket analysis, costumer support and costumer analytics, load forecasting or fraud detection. Recently, the application of data mining to the analysis of on-line communities such as MySpace or Facebook has become increasingly important.

2.1 Definition of Terminology

Not all information discovery tasks are considered to be purely data mining. Although several sections of the following areas are overlapping, there are some important distinctions between them. **Machine learning** (ML) refers to the design and development of learning algorithms that allow computers to automatically learn patterns or make decision based on data (cf. [WF05]). Machine learning is considered as a part of the data mining process (cf. Section 2.4). **Information retrieval** (IR) comprises techniques such as searching for documents or information within documents, or for metadata about documents. Moreover, IR also covers the task of searching relational databases and the World Wide Web. Although many IR tasks may involve the use of sophisticated algorithms, they rely on traditional computer science and statistical techniques. The difference between DM and IR is that IR pertains to getting back or retrieving information stored in various storage media, exactly *in the same way* it is stored, while DM aims to *reveal insight into the data* that is not quite obvious (cf. [HK06]). **OLAP** (On-line analytical process) is a decision support tool that extends traditional database query and report tools. While traditional tools describe *what* is in a database, OLAP is used to answer *why* certain things are true. OLAP consists of usually *deductive* methods that aim to verify hypothetical patters (*verification driven* mining), while DM uses the data itself to uncover such patterns (i. e., *inductive* reasoning or *discovery driven* mining). The term **data warehouse** summarizes a company-wide concept of a data-basis, where data can be stored centralized, unified and consistent. The data is detached from operative databases and can be used by various applications, for example, by DM or OLAP applications.

2.2 Connection to Other Disciplines

Even though data mining and statistics have different historical traditions, many parts of these two perspectives have converged. Many DM tasks incorporate a great deal of statistical thinking and use a variety of statistical methods. There are classification algorithms that use statistical tests during the construction of trees and rules or to validate and evaluate machine learning models. DM also adopts ideas from artificial intelligence (for example, neural networks), pattern recognition and other related fields such as optimization, information theory, signal processing or visualization techniques. Other areas can be seen as supporting areas for DM applications. For example, database technology is used to provide support for query operations, indexing or efficient storage. Figure 2.1 illustrates the interaction of DM with various related applications as well as underlying, supporting disciplines.

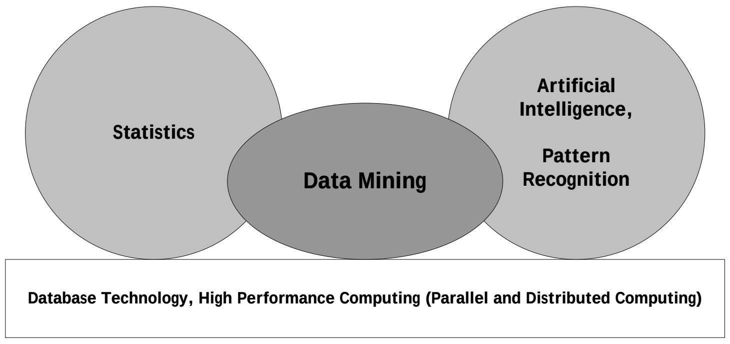
Statistics
Data Mining
Artificial Intelligence,
Pattern Recognition
Database Technology, High Performance Computing (Parallel and Distributed Computing)

Figure 2.1: Data mining and related fields (adapted from [TSK05]).

Over the last years, the tremendous improvements in techniques for collecting, storing and transferring large volumes of data have also increased the volume of data used within data mining applications, and thus have increased the demands for effective and efficient ways to deal with those large datasets. Generally, high-performance computing issues play a pivotal role in order to address the possibly large size of the data. When dealing with massive data, there are two critical dimensions to deal with: memory and runtime. Many non-incremental algorithms, where the whole data has to be loaded into the memory at once, may run out of memory space if the data is too big. On the other side, the model building process may need too much time to be performed in a feasible time period. Moreover, if there are time constraints within the prediction process, the time for assigning unclassified objects to a certain group (classification) has to be reduced. As the data grows, implementations of data mining techniques in high-performance parallel and distributed computing environments are thus becoming more and more important. Parallel and distributed algorithms are used to reduce the time complexity of the model building process and the classification process for new data as well as for several other data mining tasks such as feature reduction or clustering. To give an example, the problem could be split into smaller parts, the parts are processed independently of each other on separate processors, and finally the results are combined. Actual algorithmic implementations address a variety of parallel hardware platforms such as shared-memory systems (SMPs), distributed-memory systems (DMSs) or clusters of desktop computers or laptops. Chapter 10 focuses on parallel high-performance implementations of feature reduction techniques based on nonnegative matrix factorization (cf. Section 3.3.4).

2.3 Data

This subsection gives a brief overview of the types of data used in data mining applications. Most datasets can be viewed as a collection of data *objects* (also called *samples, records, observations, entities, instances, ...*). These data objects are described by a number of *attributes* (also called *features, variables, descriptors, ...*) that capture basic properties and characteristics of the objects. The term "variable" is often used to refer to raw, original information, whereas the terms "attribute" and "feature" are often used to refer to a variable that has been preprocessed in a certain manner.

Generally, four different types of attributes or features can be defined: *nominal*, *ordinal*, *interval* and *ratio*. The first two types belong to group of **categorical** or **qualitative** attributes which lack most of the properties of numbers. While values of *nominal* attributes are just different names and provide only enough information to distinguish one object from another (e. g., ID numbers, gender, color, . . .), *ordinal* attributes provide enough information to order objects (e. g., grades, street numbers, $\{bad, average, good\}$). The remaining two types of attributes, interval and ratio, belong to the group of **numeric** or **quantitative** attributes, which are represented by real numbers. For *interval* attributes, the differences between values are meaningful (e. g., temperature in Celsius of Fahrenheit, calendar dates, . . .), but ratios between attributes are not meaningful (e. g., 30 degrees Celsius is not twice as hot as 15 degrees Celsius). For *ratio* attributes both differences and ratios between values are meaningful (e. g., temperature in Kelvin, age, mass, length, . . .). Attributes can also be distinguished between *discrete* attributes (finite or countably infinite set of values) and *continuous* attributes (values are real number). *Binary* attributes can be considered as a special case of discrete attributes and assume only two values (yes/no, true/false,. . .).

2.4 Steps in the Knowledge Discovery Process

Several knowledge discovery process (KDP) models can be found in the literature that guide users through a sequence of steps and provide systematic approaches to successful KDD and DM. The first academic KDP models were established in the mid-1990s. The main emphasis of these KDP models was to provide a sequence of activities that helps to execute a KDP in an arbitrary domain. Industrial models followed a few years later, with more emphasis on commercial aspects. The academic process model following Fayyad *et al.* [FPS96] – one of the first, and probably the most popular and most cited model – is an interactive and iterative model that consists of nine steps (with possible loops between any two steps) involving decisions made by the user, but lacks the specific details of the feedback loops. Contrary to this sequential model, the business oriented CRISP-DM model (cross-industry standard process for data mining) emphasizes the cyclical character of data mining, and contains loops between several of its six steps. A detailed description of this (mainly) industrial model (which has also been incorporated into a commercial KDD system called *clementine*[1]) can be found in [CCK00]. Another representative industrial model based on five steps can be found in [CHS97], while [AB98] provide an eight-step academic model. Moreover, Cios *et al.* [CPS07] published a hybrid academic/industrial six-step model which is mainly based on the CRIPS model, but provides a more general, research-orientated description of the steps and introduces

[1]`http:/www.spss.com/clementine`

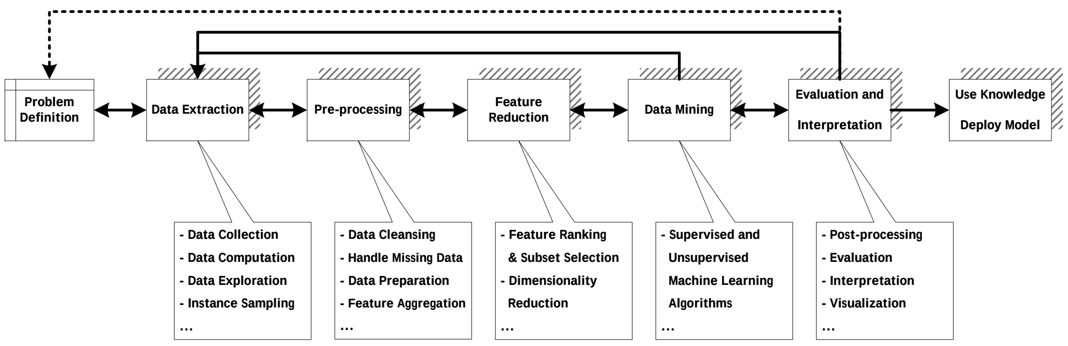

Figure 2.2: Steps in the KDD process.

some new explicit feedback mechanisms. [CPS07] also provide a detailed comparison of all five process models mentioned above.

In this thesis we introduce a knowledge discovery process model that is strongly correlated with the ones introduced by [FPS96] and [CPS07], but puts more emphasis on technical aspects and the technical descriptions of each step than on the problem definition steps. Contrary to the model by Cios *et al.* [CPS07], we assume that the objectives and requirements from a business or from an academic perspective are already known and that the data mining goals have already been determined, i. e., the problem definition step is already finished. Unlike most other models, we split up the data preparation step into three autonomous steps (steps 1-3), each with emphasis on a certain aspect. The first step – the data extraction step – deals with collecting and computing data as well as exploring the data objects and selecting subsets of them. The next step comprises simple pre-processing tasks such as data cleansing, handling missing data, normalization, discretization or feature aggregation. It is important to notice that more sophisticated methods for feature reduction are outsourced into a separate step, the feature reduction step. In other KDP models, simple pre-processing tasks and sophisticated methods for reducing the feature space are usually pooled together in one step. In our model, this third step comprises several techniques for ranking or selecting subsets of original features, or for reducing the dimensionality of the feature space due to dimensionality reduction or feature transformation methods. The subsequent data mining and evaluation steps are mainly similar to the KDP model introduced by Cios *et al.* [CPS07]. Our KDP model is depicted in Figure 2.2 and shows the five main steps mentioned before.

Feedback loops. As can be seen from the model, there are loops between any two steps, as well as three explicit feedback loops from the data mining step and the evaluation and interpretation step backwards to the data extraction step as well as from the evaluation step backwards to the problem definition. The most common reasons for the feedback loop from *evaluation and interpretation* to *problem definition* is poor understanding of the data, requiring modification of the project's goal. Incorrect understanding and interpretation of the domain and incorrect design or understanding of the problem restrictions and requirements may require a repetition of the compete KDP. The feedback loops from both *data mining* and *evaluation and interpretation* back to *data extraction* are ambiguous. On the one hand, if the results are unsatisfactory there might be a need for better understanding or better selection of the data to avoid subsequent failures in the KDP process. On the other hand, if the results are satisfactory, the feedback loops may be used to provide important information for this step for subsequent runs of the KDP (e. g., only a specific subset of objects or features might be required, thus saving memory requirements and runtime).

The tasks and techniques associated with each of the five steps following the problem definition are described on the next pages. Moreover, for each step, possible feedback loops back to preceding steps are discussed. For better readability, the detailed descriptions of the techniques involved in steps 3 and 4 – feature reduction and data mining – are both summarized in separate chapters (Chapters 3 and 4).

2.4.1 Step 1 – Data Extraction

The first task in the data extraction phase is to create the target dataset. This extraction can be performed in several ways. In the easiest setting, the data has already been extracted and is stored in files, or can easily be collected from a database (e. g., using SQL-queries) or from a data warehouse (e. g., using OLAP tools, cf. Section 2.1). In other scenarios, there might not be any extracted information about the data, but instead samples are stored in their original format and the features and properties of the data have to be computed explicitly. For example, consider a set of email messages saved in some email format (e. g., mbox-format) that should be used to create a learning model that is able to classify newly arriving email messages to one of the groups spam/not-spam. In this case, features describing these email messages have to be computed in order to use this sample for building a learning model and classifying unlabeled email messages. If the data are not already available, the data extraction phase might take a large part of the time and effort of the entire knowledge discovery process.

At this point it is also very important to get to know the data. Data exploration is the preliminary investigation of the data in order to better understand its specific characteristics. Summary statistics (mean, standard deviation, . . .), visualization techniques (histograms, scatter plots, . . .) and OLAP techniques are useful tools to explore the dataset at an early stage and to prevent (costly) failures at later steps.

Instance sampling is the process of selecting a representative subset of data objects (not features!) in order to reduce the number of objects within a dataset – ideally without loss in the mining quality. The selected subsets can be chosen randomly (either with or without replacement) or via sampling strategies that sample objects from pre-defined groups or clusters (stratified sampling). In other settings, the focus might be on the analysis of a specific group of samples, where a systematic selection of samples having similar properties in some sense is performed. Usually, the selection of subsets of features (see step 3, *feature reduction*) is a more complicated and complex task than the selection of samples of objects and has to be performed carefully, since a lot of information about the data might be lost if the wrong features are removed. The feedback loop from this step to *problem definition* is based on the need for additional domain knowledge in order for better understanding the data.

2.4.2 Step 2 – Data Pre-processing

It is unrealistic to expect that data will be perfect after they have been extracted. Since good models usually need good data, a thorough cleansing of the data is an important step to improve the quality of data mining methods. Not only the correctness, also the consistency of values is important. Missing data can also be a particularly pernicious problem. Especially when the dataset is small or the number of missing fields is large, not all records with a missing field can be deleted from the sample. Moreover, the fact that a value is missing may be significant itself. A widely applied approach is to calculate a substitute value for missing fields, for example, the median or mean of a variable. Furthermore, several data mining approaches (especially many clustering approaches, but also some learning methods) can be modified to ignore missing values.

Data preparation consists of techniques such as standardizing or z-scoring (shifting the mean of a variable to 0 and the standard deviation to 1) or normalizing the data (scaling the range of an attribute to some pre-defined interval, for example [0,1]). It is also sometimes necessary to transform a continuous attribute into a categorical attribute (discretization), or both continuous and discrete attributes may need to be transformed into one or more binary attributes (binarization).

Feature aggregation aims at combining two or more variables into a single feature. Depending on the type of the attributes (qualitative, quantitative, etc.; cf. Section 2.3) the aggregate transaction is created differently. Since feature aggregation is usually not an automated feature reduction process but requires good knowledge of the data, we assign it to the data pre-processing step and not to the feature reduction step.

The feedback loop to *data extraction* is especially important if the quality of the data is unsatisfactory, for example due to noisy data, too many missing values, or missing heterogeneity among features. In the data pre-processing step, the quality of the data is analyzed using simple statistical methods such as mean, variance, or correlation between features. In some cases, there might also be a need for additional information or more specific information about the data.

2.4.3 Step 3 – Feature Reduction

In many application areas, datasets can have a very large number of features. As the dimensionality of the feature space increases, many types of data analysis and classification become significantly harder, and, additionally, the data becomes increasingly sparse in the space it occupies which can lead to big difficulties for both supervised and unsupervised learning. In the literature, this phenomenon is referred to as the *curse of dimensionality* and was first mentioned 1961 by Bellman [Bel61]. On the one hand, in the case of supervised learning or classification, there might be too few data objects to allow the creation of a reliable model that assigns a class to all possible objects. On the other hand, for unsupervised learning methods or clustering algorithms, various vitally important definitions (e. g., density or distance between points) may become less meaningful (cf. [Pow07]). For example, the proximity of points tends to become more uniform in high-dimensional data, and, moreover, Tan *et al.* [TSK05] state that the traditional Euclidean notion of density, which is the number of points per unit volume, becomes meaningless. As a result, a high number of features can lead to lower classification accuracy and clusters of poor quality.

Besides this key factor, Tan *et al.* [TSK05] mention various other benefits from reducing the number of features. Such a reduction can (*i*) lead to a better understandable model, (*ii*) simplify the usage of different visualization techniques (e. g., looking at pairs or triplets of attributes, self-organizing maps [Koh00] (cf. 2.4.4), etc.), and, last but not least, (*iii*) significantly reduce the computational cost and memory requirements of the classification algorithm applied on the data.

Overfitting. With increasing number of attributes, a classification algorithm will usually be able to fit the training data better. A classification algorithm is said to overfit relative to a simpler algorithm (the same method, but with a smaller number of attributes) if it is more accurate in fitting training data but less accurate in classifying (predicting) new data (cf., for example, [TSK05]). This approach is in accordance with Occam's razor: *simpler models are preferred over more complex ones* (cf., for example, [Hir97]). Especially in the case where training examples are rare (compared to the number of features), the learning algorithm may adjust to very specific random features of the training data that do not have a causal relation to the target function (i. e., they are not good at predicting unlabeled objects). Classification models that are based on a small number of training samples are also susceptible to overfitting, even when the number of attributes is small.

Generally, feature reduction strategies can be grouped into two main groups – *feature selection* (FS) and *dimensionality reduction* (DR). The terminology used in this thesis follows the work of Tan *et al.* [TSK05], but other (differing and thus often confusing) terminology is widely used throughout the literature. The main idea of feature selection strategies is to remove redundant or irrelevant features from the dataset as they can lead to a reduction of the classification accuracy or clustering quality and to an unnecessary increase of computational cost (cf. [KS96, BL97]). The advantage of feature selection (over DR) is that no information about the importance of single features is lost. On the other hand, if a small set of features is required and the original features are very diverse, information may be lost as some of the features must be omitted.

With dimensionality reduction techniques the size of the attribute space can often be decreased strikingly without loosing a lot of information of the original attribute space. An important disadvantage of dimensionality reduction is the fact that the linear combinations of the original features are usually not easily interpretable and the information about how much an original attribute contributes is often lost.

The feedback loop from this step to *data pre-processing* is needed if the properties of the data (e. g., discretized, normalized data) are not consistent with the applied feature reduction or data mining techniques, i. e., a feature reduction method that is based on discrete data might be unable to work with continuous data. In this case, the data has to be discretized in step 2 prior to the application of the feature reduction method.

Both feature selection and dimensionality reduction techniques are an important part of this thesis and are discussed in more detail in Chapter 3.

2.4.4 Step 4 – Data Mining

At the core of the knowledge discovery process is the application of specific data mining methods. Typically, data mining methods have two high-level goals: prediction and/or description. The goal of *predictive* data mining tasks (in the literature also referred to as *supervised learning*) is to build a model that predicts unknown (or *future*) values of a particular attribute of interest based on known values of some other attributes. The attribute to be predicted is commonly known as target, class or dependent variable. Attributes used for making the prediction are known as the explanatory or independent variables (cf. [TSK05]).

The task of building a model for the target variable as function of the explanatory variables is called *predictive modeling*. Classification (prediction of discrete target variables) and regression (prediction of continuous target variables) are the two main types of predictive modeling. The goal of classification and regression is to build a model that minimizes the error between the predicted and true values of the target variable. In almost any data mining application, the choice of which specific learning algorithm should be used is a critical step, since there is no generally "best" learning algorithm (a statement, which can be found in almost all reviews and books about data mining). One algorithm may be more accurate in the sense of classifying unlabeled objects, but very slow in performance. Other algorithms may be very fast but not as accurate as others. In Chapter 4, several supervised learning algorithms are reviewed and compared in terms of expected classification accuracy, comprehensibility, complexity and speed.

The process of building predictive models requires a well-defined training and test set in order to insure that the data is trained on one dataset and tested on another dataset with different objects in order to measure the expected generalized error. This results in an estimate of how the model will perform on future data that are similar to the training and test data. Sometimes a third set, called validation dataset, is used to act as a further independent measure of the model's accuracy. There are several methods that are commonly used to split the data samples into training and test set. In the *holdout method*, the data with labeled examples is partitioned into two disjoint sets (training and test set). The classification model is then built on the training set and its performance is evaluated on the test set. The proportion of the size of the training and test set depends on the sample size, the learning algorithm and the application area. In the *random sub-sampling* approach, the holdout method is repeated several times with randomly selected training and test sets (sampling without replacement). *Cross-validation* is a popular and commonly used approach, where each object is used the same number of times for training and exactly once for testing. For an n-fold cross-validation each sample is split randomly into n parts of roughly equal size and alternately n-1 of these parts are used for training and one of them for testing. The *leave-one-out* approach is a special case of cross-validation where the test set contains only one object. The advantage of cross-validation is that all data is used for training, but due to its nature this approach may be computationally expensive. Contrary to the approaches mentioned so far, the *bootstrap approach* samples objects with replacement, i. e., an object already chosen for the training set is put back and might be redrawn such that it appears twice or even more often in the training set. The objects not chosen for the training set are used as test set.

Contrary to predictive tasks, *descriptive* data mining tasks (also referred to as ***un****supervised learning*) start with a number of objects and try to define similarity among them with respect to a number of attributes. Here, the goal is to derive patterns (clusters, anomalies, correlations, . . .) that summarize the underlying relationships in the data in human-understandable terms. As the name implies, unsupervised classification methods work without the class information of a sample, i. e., there is no a priori known output at the time of classification. All attributes are treated in the same way – there is no distinction between explanatory and dependent variables.

Clustering or cluster analysis is one of the core descriptive data mining tasks. Here, closely related observations are assigned into subsets (called clusters) so that observations in the same cluster are similar in some sense. Clustering algorithms can typically be divided into hierarchical and partitional clustering. While hierarchical algorithms find successive clusters using previously established clusters (using a bottom-up or a top-down strategy), partitional algorithms determine all clusters at once. The most famous clustering method is the (partitional) k-means algorithm [JD88], which assigns each point to one of the k clusters whose center is nearest. The center is the average (typically the arithmetic mean) of all points in the cluster. Famous application areas for clustering algorithms are document clustering, anomaly detection, product positioning, or gene sequence analysis. Another classical example of unsupervised learning methods are self-organizing maps (SOMs, cf. Section 4.9), which are special types of neural networks that generate a topology-preserving non-linear mapping of a high-dimensional input space to a low-dimensional space.

The algorithms used in Part II of this thesis are purely predictive algorithms (i. e., supervised learning). As already mentioned, several classification algorithms (including all algorithms used in Part II of this thesis) are described in more detail in Chapter 4.

The feedback loop from this step (*data mining*) to *feature reduction* is one of the most often executed feedback steps, as these two steps (feature reduction and data mining) are tightly coupled. This is due to the fact that almost all feature selection and dimensionality reduction techniques need data mining methods to evaluate the quality of a specific feature set. Some of these methods (wrapper methods, see Section 3.2.2, as well as embedded methods, see Section 3.2.3) in fact use a learning algorithm as an integral part of the feature selection process.

Table 2.1: Confusion matrix for a two class problem.

		Predicted class	
		yes	**no**
Actual class	**yes**	*true positive (TP)*	*false negative (FN)*
	no	*false positive (FP)*	*true negative (TN)*

2.4.5 Step 5 – Post-processing and Interpretation

Once a classification model has been constructed, it can be used to predict the class of previously unseen objects. In a simple binary case with classes "yes" (positives) and "no" (negatives), a single prediction has four different possible outcomes (shown in a confusion matrix in Table 2.1). The *true positives (TP)* and *true negatives (TN)* are correct classifications, i. e., an object of class "yes" is correctly predicted as "yes" (analogously for class "no"). When the outcome of an object of class "no" is incorrectly predicted as "yes" (positive), this is called a *false positive (FP)* (incorrectly predicted as positive), and a *false negative (FN)* occurs when an object is incorrectly predicted as negative ("no") when it actually belongs to class "yes". In statistics a FP is often referred to as *type I error* and a FN as *type II error.* In many cases the two kinds of error (FP and FN) will have different costs. For example, the cost of classifying a legitimate email message as spam (loss of possibly important information) may be much higher than the cost of classifying a spam message as not-spam (cf. Section 5.1.2). Based on these four possible predictions, various performance metrics can be computed to compare the quality of the classification results of a two class problem. The most important performance metrics are summarized in the following.

- The *overall accuracy* or *aggregated accuracy* or *overall success rate* is the total number of correct classifications (which is $\mathrm{TP}+\mathrm{TN}$) divided by the total number of classifications (which is $\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}$). Throughout the literature, varying terminologies such as *predictive power* or *generalization error* are used to refer to the general or overall accuracy of learning algorithms.

- The *true positive rate* is TP divided by the total number of positives (which is $\mathrm{TP}+\mathrm{FN}$), and the *false positive rate* is FP divided by the total number of negatives (which is $\mathrm{FP}+\mathrm{TN}$).

- Similarly, the *true negative rate* is TN divided by the total number of negatives

Table 2.2: Confusion matrix for a three class problem.

		Predicted class		
		a	**b**	**c**
Actual class	**a**	*(a pred. as a)*	(a pred. as b)	(a pred. as c)
	b	(b pred. as a)	*(b pred. as b)*	(b pred. as c)
	c	(c pred. as a)	(c pred. as b)	*(c pred. as c)*

(which is the same as 1 - FP rate), and the *false negative rate* is FN divided by the total number of positives (which is the same as 1 - TP rate).

- In some application areas, the true positive rate is also referred to as *sensitivity*, and the true negative rate is referred to as *specificity*. Moreover, recall and precision are also two widely used performance metrics.

- *Recall* measures the fraction of positive compounds correctly predicted by the classifier and is thus identically to sensitivity. *Precision* (also called *hit rate*) determines the fraction of TP in the group which the classifier has declared as a positive class (which is $\mathrm{TP} + \mathrm{FP}$).

In multi-class prediction (i. e., more than two classes), the confusion matrix can be extended to a matrix with rows and columns for each class. Each matrix element shows the number of test examples for which the actual class is the row and the predicted class is the column. Table 2.2 shows an example of a confusion matrix for a three class problem. All correct classifications are highlighted in italic format.

Since different classifiers usually show a different classification performance on different datasets, it is often useful to compare the performance of different classifiers to determine which classifier works better on a given dataset. Besides a comparison of the overall accuracy, TP rate, FP rate, sensitivity, specificity and precision, a statistical t-test can be used to check the null hypothesis that the mean difference between two classifiers is zero. In the case of unsupervised learning (clustering algorithms, SOMs, ...), the evaluation of the learning algorithm is often a difficult task, since no information about the class labels are available (i. e., it might be impossible to validate the outcome of the learning algorithm with the metrics mentioned above). Instead, measures such as intra-cluster similarity (validation if instances within a cluster are similar) and inter-cluster similarity (validation if instances from different clusters are dissimilar) can be applied.

Visualizing the outcome helps to summarize results in a compact way that is often easier to interpret than numbers and tables. Besides plotting curves for TP, TN, FP, FN, recall, precision and the overall classification accuracy, receiver operating characteristic curves (ROC), a graphical plot of the sensitivity vs. (1 - specificity) for a binary classification, are commonly used visualization techniques for supervised classification models.

At the end of the KDP is the utilization and deployment of the learning model. Once a classification model is built correctly and validated, it can either be used directly by viewing the model and its results or it can be applied to different data. In the latter case, the model often needs to be incorporated into a program using an API or code generated by some specific data mining tool.

2.5 Relevant Literature

Since the number of references for this introductory chapter is rather large, all references are summarized in this section. In the subsequent chapters, relevant literature will be included directly in the main text.

A vast amount of books and survey papers exist which present a detailed overview of various data mining and knowledge discovery techniques and methods. Beside numerous introductory textbooks [Dun02, TSK05, HK06, CPS07, Lar07, ML07], there are books which focus more on technical aspects of machine learning algorithms [WF05], or put more emphasis on statistical learning theory [Vap99, HTF02]. A discussion of the overfitting problem can be found, for example, in [BFS84] and [Sch93].

Some data mining and machine learning books focus on a specific application area, for example, web data mining [Liu07], text mining [FS06], or pattern recognition [DHS01, Bis07]. Other books focus on the application of machine learning and knowledge discovery methods in bio- and pharmacoinformatics [WZT04, HP07, ZZL08], QSAR and molecular modeling [Lea01, HSR08, Gup09], or put their emphasis more on the business application of data mining methods [BL04].

The work by Kargupta *et al.* [KC00, KJS04, KHY08] focuses especially on distributed aspects of data mining and knowledge discovery applications, while [ZH00] and [GKK03] discuss the parallelization potential of data mining algorithms. More related work for distributed and parallel data mining as well as high-performance computing aspects of data mining applications can be found in Chapter 10.

Several differing knowledge discovery process (KDP) models are described in the literature, for example in [FPS96, CHS97, AB98, CCK00, CPS07], and a comparison of these models can be found in [CPS07]. Descriptions of data extraction and data pre-processing can be found in almost all introductory data mining textbooks. Relevant literature for feature reduction methods is discussed in Chapter 3, and literature dealing with specific supervised learning algorithms can be found in Chapter 4. An empirical study to compare the performance metrics obtained using different estimation methods such as random sub-sampling, bootstrapping and cross-validation can be found in [Koh95].

A classical introduction to clustering techniques for statistics can be found, for example, in [KR05], and several clustering techniques containing both hierarchical and partitional relocation clustering are surveyed in [Ber02]. Detailed information about approximate statistical tests for comparing classification algorithms can be found in [Die98], and information for estimating confidence intervals for accuracy measures can be found in [TSK05]. Some comments on the definition of the generalization error can be found in [Sch03], and several evaluation possibilities for clustering techniques are summarized in [DGG07]. Visualization techniques in knowledge discovery and data mining are discussed in [Spe00] and [FGW01].

Chapter 3

Feature Reduction

The *curse of dimensionality*, introduced in Section 2.4.3, is based on the fact that high dimensional data is often difficult to work with. A large number of features can increase the noise of the data and thus the error of a learning algorithm, especially if there are not enough observations to get good estimates. Moreover, the effort and computational cost of data mining applications grow with increasing dimension. As already mentioned in Section 2.4.3, there are several methods that aim at solving these problems by reducing the number of features and thus the dimensionality of the data. In this section, these methods are categorized into two main groups and several of them are discussed and compared to each other.

The general objectives for reducing the number of features are to avoid overfitting (cf. Section 2.4.4) and thus to improve the prediction performance of classification algorithms, reduce the computational cost of the training and prediction phase, and also to provide a better understanding of the achieved results and gain a deeper insight into the underlying processes that generated the data. Since the terminology used in the area of feature reduction methods varies throughout the literature, this work follows the notation of Tan *et al.* [TSK05] for the distinction between *feature selection* (reduction of the dimensionality by *selecting* attributes that are a *subset* of the old, original variables) and *dimensionality reduction* (reduction of the dimensionality by using low-rank approximation techniques to *create new* attributes that are a *combination* of the old, original variables). In other literature, dimensionality reduction is sometimes also referred to as feature transformation, feature extraction or feature construction. In this thesis, the term *feature reduction* is used as a general term comprising both, feature selection and dimensionality reduction methods.

3.1 Relevant Literature

Contrary to Chapter 2, in this chapter relevant literature for feature selection and dimensionality reduction and methods is included directly in the main text.

3.2 Feature Selection

Feature selection (FS) refers to selecting attributes that are a subset of the original attributes. According to class information availability in the data, there are supervised and unsupervised (cf. [Dy07]) feature selection methods. The focus of this thesis is on supervised feature selection methods. In this case, feature selection can be defined as follows:

Definition 1 (Supervised feature selection)**.** Given a set of features $F = \{f_1, \ldots, f_i, \ldots, f_n\}$, find a subset $F' \subseteq F$ that maximizes the ability of the learning algorithm to classify patterns. Formally, F' should maximize some scoring function Θ, such that $F' = arg\,max_{G \in \Gamma}\{\Theta(G)\}$, where Γ is the space of all possible feature subsets of F.

Feature selection techniques do not alter the original representations of features, but select a subset of them. Hence, these techniques preserve the original semantics of the features, offering the advantage of interpretability by a domain expert. In theory, the goal is to find the optimal feature subset that maximizes the scoring function above. The selection of features (or subsets of features) should be performed on the training set only, the test set is then used to validate the quality of the selected features (subsets).

Throughout the literature (cf., for example, [KJ97, GE03, YL03, LY05b, SIL07]), feature subset selection approaches are categorized into three main groups: *filter methods*, *wrapper methods* and *embedded approaches.* Filter methods rely on general characteristics of the training data to evaluate and select subsets of features without involving a learning algorithm. Contrary to that, wrapper approaches use a classification algorithm as a black box to assess the prediction accuracy of various subsets. The last group, embedded approaches, performs the feature selection process as an integral part of the machine learning algorithm. In the following, these three techniques are described in detail. A visual overview of these three approaches is given in Figure 3.1 (**A** = filter methods, **B** = wrapper methods, **C** = embedded approaches).

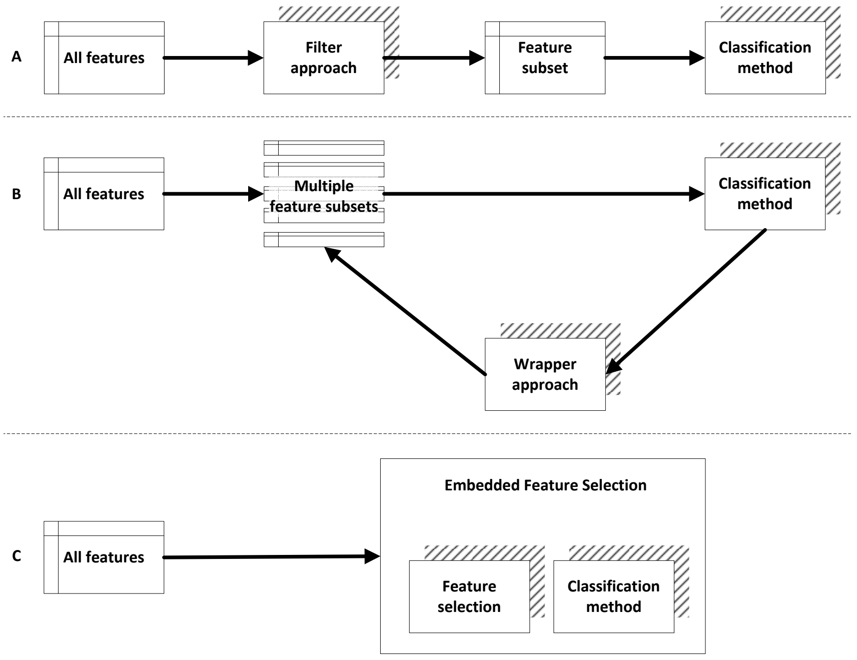

Figure 3.1: A simple scheme of feature selection techniques.

3.2.1 Filter Methods

Filter methods are classifier agnostic, no-feedback, pre-selection methods that are independent of the machine learning algorithm to be applied. Following the classification of [SIL07] (which slightly differs from the classification in [KJ97] and [GE03]), filter methods can further be divided into *univariate* and *multivariate* techniques. Univariate filter models consider one feature at a time, while multivariate methods consider subsets of features together, aiming at incorporating feature dependencies. In the survey by Guyon and Elisseeff [GE03], univariate filter methods are referred to as *single variable classifiers*, and multivariate filter methods are grouped together with wrapper methods and embedded methods and referred to as *variable subset selection* methods.

Univariate filter methods. These methods consider features separately and usually make use of some scoring function to assign weights to features individually and rank them based on their relevance to the target concept (cf., for example, [YL03]). In the literature, this procedure is commonly known as *feature ranking* or *feature weighting*. A feature will be selected if its weight or relevance is greater than some threshold value.

Definition 2 (Feature ranking)**.** Given a set of features $F = \{f_1, \dots, f_i, \dots, f_n\}$, order the features by the value of some individual scoring function $S(f)$. If $S(f_i)$ is greater than a threshold value t, feature f_i is added to the new feature subset F'.

It may happen that features in this subset contain redundant information or do not provide information by themselves (without being combined with other features). For example, perfectly correlated features are truly redundant in the sense that no additional information is gained/lost by adding/removing them to/from the subset. These features are often a problem for univariate filter methods. Univariate filter methods may select several redundant features containing the same information, because only the individual feature weights are considered. On the other side, a feature that is useless by itself can provide useful information when taken in combination with others. These features might be excluded from the feature subset because their individual predictive power is low.

Various univariate filter methods exist, such as the family of instance-based RelieF algorithms, or statistical approaches such as Pearson's correlation, linear regression or χ^2 statistics.

RelieF (recursive elimination of features) algorithms (cf. [KR92]) are able to detect conditional dependencies of features between instances. They evaluate the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class. Most RelieF algorithms can operate on both discrete and continuous class data. A theoretical and empirical analysis of several ReliefF algorithms can be found in [RK03].

The *Pearson's product moment correlation* (*Pearson's correlation* for short, cf. [Guy08]) reflects the degree of linear relationship between two variables, and is defined as $R(i) = \frac{cov(\mathbf{x}_i,\mathbf{y})}{\sqrt{var(\mathbf{x}_i)var(\mathbf{y})}}$. In *linear regression* models (cf., for example, [HL95]), $R(i)^2$ is used as a variable ranking criterion and enforces a ranking according to goodness of linear fit of individual variables (cf. [GE03]). The *χ^2-statistics method* (cf. [LS96]) measures the independence between explanatory and target variables and evaluates the worth of an attribute by computing the value of the χ^2-statistic with respect to the class.

Besides instance-based and statistical approaches, there are several univariate filter methods that use *information theoretic criteria* for variable selection, such as information gain (also called Kullback-Leibler divergence) and gain ratio, both described in [Qui93, WF05, Bra07] and summarized briefly in the following.

Information gain. One option for ranking features of a given dataset according to how well they differentiate the classes of objects is to use their information gain (IG), which is also used to compute splitting criteria for some decision tree algorithms (for example, the C4.5 algorithm discussed in Section 4.3). Information gain is based on *entropy*, an information-theoretic measure of the "uncertainty" contained in a training set, due to more than one possible classifications (cf. [Bra07]). Given a discrete explanatory attribute $\mathbf{x}$ with respect to the (also discrete) class (or target) attribute $\mathbf{y}$, the uncertainty about the value of $\mathbf{y}$ is defined as the overall entropy

$$H(\mathbf{y}) := -\sum_{i=1}^{k} P(\mathbf{y} = y_i) \log_2 \left(P(\mathbf{y} = y_i)\right). \tag{3.1}$$

The conditional entropy of $\mathbf{y}$ given $\mathbf{x}$ is then defined as

$$H(\mathbf{y} \,|\, \mathbf{x}) := -\sum_{j=1}^{l} P(\mathbf{x} = x_i)\, H\left(\mathbf{y} \,|\, \mathbf{x} = x_i\right). \tag{3.2}$$

The reduction in entropy ("uncertainty") or the "gain in information" of each attribute $\mathbf{x}$ is then computed as

$$IG(\mathbf{y}; \mathbf{x}) := H(\mathbf{y}) - H(\mathbf{y} \,|\, \mathbf{x}). \tag{3.3}$$

Example: Given a training set of 14 instances, a binary classification problem (with classes "yes" and "no"), and a variable "temperature" with three discrete values $\{$*hot, mild, cool*$\}$. For five instances of the training sample, the value of the variable temperature is *hot*. Out of those five instances, the class variable of two instances is "yes", and the class variable of the three other instances is "no". The variable temperature is *mild* for four instances of the training set, all of them have the class variable "yes". Finally, the value of the variable temperature of the remaining five instances is *cold*, three of them belong to class "yes", two of them to class "no". Overall, nine out of 14 instances belong to the class "yes", and five instances belong to the class "no".

Thus, the overall gain in information ("information gain") for the attribute *temperature* can be computed as $IG(temperature) := H([9,5]) - H([2,3],[4,0],[3,2])$, where $H([9,5])$ can be computed as $-9/14 \times log_2\, 9/14 - 5/14 \times log_2\, 5/14 = 0.940$. Moreover, $H([2,3]) \equiv H([3,2]) = 0.971$, and $H(4,0) = 0$. Thus, $H([2,3],[4,0],[3,2])$ $= (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971 = 0.693$. The information gain for the variable temperature then becomes $IG(temperature) = 0.940 - 0.693 = 0.247$. A high information gain indicates that the variable is able to differentiate well between different classes of objects.

Gain ratio. Information gain favors features which assume many *different* values. Since this property of a feature is not necessarily connected with the splitting information of a feature, features can also be ranked based on their information gain ratio (GR), which normalizes the information gain and is defined as

$$GR(\mathbf{y}; \mathbf{x}) := IG(\mathbf{y}; \mathbf{x}) / \text{splitinfo}(\mathbf{x}), \tag{3.4}$$

where

$$\text{splitinfo}(\mathbf{x}) := -\sum_{i=1}^{l} P(\mathbf{x} = x_i)\ \log_2\ P(\mathbf{x} = x_i). \tag{3.5}$$

Although feature ranking may not be optimal, it may be preferable to other feature reduction methods because of its computational and statistical scalability. Feature ranking is computationally efficient, because it requires only the computation of n scores and sorting the scores, and statistically robust, because it is known to avoid overfitting due to its low variance compared to multivariate feature selection methods (cf. [HTF02]). Nevertheless, since only the individual predictive power of features is taken into account and feature dependencies are ignored, many redundant features containing similar information may be chosen.

Multivariate filter methods. Members of this second group of filter methods (also referred to as *subset search evaluation*) search through candidate feature subsets guided by a certain evaluation measure which captures the quality of each *subset* (not only the individual predictive power of single features). Contrary to wrapper methods, multivariate filter methods do *not* rely on a specific learning algorithm. Instead, consistency measures or correlation measures are often used to find a minimum number of features that are able to separate classes as good as the full set of features can. An overview over several subset search evaluation methods can be found in [LM98, DLM00, DL03]. Subset search evaluation can be defined as follows:

Definition 3 (Subset search evaluation)**.** Given a set of features $F = \{f_1, \ldots, f_i, \ldots, f_n\}$, find a feature subset F' such that the classification accuracy based on the feature subset F' is maximized.

The "best" subset search evaluation approach to find the optimal subset would be to try all possible feature subsets as input for the classification algorithm and choose that subset which leads to the best results (in terms of classification accuracy). Since the number of possible attribute subsets grows exponentially with the number of attributes this exhaustive search method is impractical for all but simple toy problems. To illustrate this, Figure 3.2 shows all 16 (2^4) possible feature subsets for a very simple four-feature problem. Generally, the size of the search space for n features is $O(2^n)$. Hence, most multivariate filter methods as well as most wrapper methods use heuristic search methods to balance the quality of the subset and the cost of finding it.

Famous examples of multivariate filter methods are correlation-based feature selection (CBFS, cf. [Hal00]) and its extension, fast CBFS (cf. [YL04]). Generally

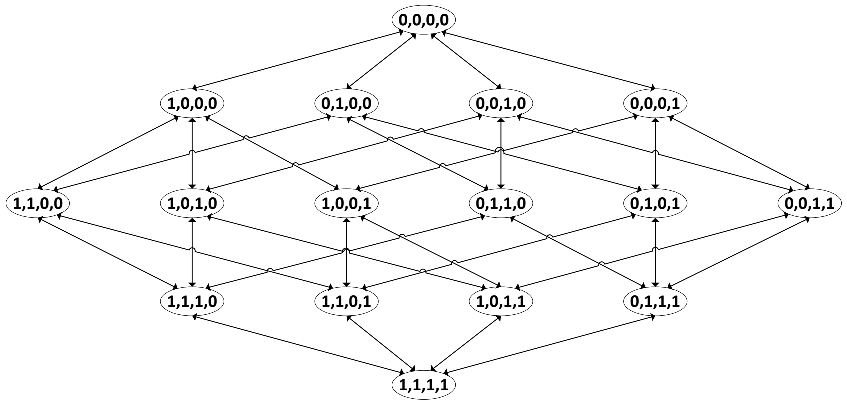

Figure 3.2: All 2^n possible feature subsets for a simple four-feature problem [KJ97], "1" and "0" denotes the appearance/absence of a feature in a subset, respectively. Each node is connected to nodes that have one feature deleted or added.

speaking, the idea behind CBFS is that a feature is important if it is relevant to the class concept but is not redundant to any of the other relevant features. Applied with correlation, the goodness of a feature is measured, i. e., it is measured whether a feature is highly correlated with the class but not highly correlated with any of the other features. In Fleuret's approach [Fle04], conditional mutual information is used to catch dependencies among features. The author states that by picking features which maximize their mutual information with the class to predict conditional to any feature already picked, the method ensures the selection of features which are both individually informative and two-by-two weakly dependent. Experimental results show the strength of this approach, even when combined with a very simple naïve Bayesian classifier (cf. Section 4.9). Moreover, self-organizing or *Kohonen* maps (SOMs, cf. Section 4.9), which are usually used as unsupervised classification methods (cf. Section 2.4.4), can also be applied independently of the classification to the task of subset search evaluation. Usually, SOMs generate a topology-preserving non-linear mapping of a high-dimensional input space (the *descriptor space*) to a low-dimensional space, but they can also be applied as feature selection methods by identifying correlations between features (cf., for example, the recent study in [KOS09]).

Compared to feature ranking methods, multivariate filter algorithms are able to identify and remove redundancies between features. Concerning the computational cost, since multivariate filter algorithms need to measure the quality of a possibly large number of candidate feature subsets instead of the quality of single features, they are less scalable and slower than univariate techniques (cf. [GE03, SIL07]). However, they still have a better computational complexity than wrapper methods.

3.2.2 Wrapper Methods

Wrapper methods (cf. [KJ97, Das01]) are feedback methods which incorporate the machine learning algorithm in the feature selection process, i. e., they rely on the performance of a specific classifier to evaluate the quality of a set of features. Here, the classification algorithm is used as a black box (cf. [KJ97]). Wrapper methods search through the space of feature subsets using a learning algorithm to guide the search. To search the space of different feature subsets, a search algorithm is "wrapped" around the classification model. A search procedure in the space of possible feature subsets is defined, various subsets of features are generated, and the estimated classification accuracy of the learning algorithm for each feature subset is evaluated.

Generally, wrapper methods can be divided into two groups, deterministic and randomized methods.

Deterministic wrapper methods. These methods search through the space of available feature either forwards or backwards. In the forward selection process, single attributes are added to an initially empty set of attributes. At each step, variables that are not already in the model are tested for inclusion in the model and the most significant variables (e. g., the variable that increases the classification accuracy based on the feature subset the most) are added to the set of attributes. In the forward stage-wise selection technique only one variable can be added to the set at one step. For example, in the best first search method the space of possible attribute subsets is searched by a greedy hillclimbing algorithm augmented with a backtracking facility. In the backward elimination process one starts with the full set of features and deletes attributes one at a time (cf. [AB95, WF05]). In a study by Guyon and Elisseeff (cf. [GE03]), a recursive implementation of this technique called recursive feature elimination (RFE) has successfully been applied to the task of gene selection by utilizing support vector machines (cf. Section 4.5) as feature ranking method. A paired t-test is often used to compute the probability if other subsets may perform substantially better. If this probability is lower than a pre-defined threshold the search is stopped. The result is a (heuristically) optimal feature subset for the applied learning algorithm.

Li and Yang [LY05a] have compared multiple classifiers with FS in recursive and non-recursive settings and showed that the ability of a classifier for penalizing redundant features and promoting independent features in the recursive process has a strong influence on its success.

Randomized (non-deterministic) wrapper methods. Compared to deterministic wrapper methods, randomized wrapper algorithms search the next feature subset partly at random (i. e., the current subset does not directly grow or shrink from any previous set following a deterministic rule). Single features or several features at once can be added, removed, or replaced from the previous feature set. Famous representatives of randomized wrapper methods are *genetic algorithms* (cf. [Gol89] or [Vos99]), stochastic search methods which are inspired by evolutionary biology and use techniques encouraged from evolutionary processes such as mutation, crossover, selection and inheritance to select a feature subset. The application of genetic algorithms as non-deterministic wrapper methods has been studied, for example, in [JC99] or [HSL02]. *Simulated annealing* (cf. [KGV83]) – another stochastic search method – is a generic probabilistic global search meta-algorithm for the global optimization problem, which has also been successfully applied as wrapper method (cf. [BZF06]).

The interaction of wrapper methods with the classification algorithm often results in better classification accuracy of the selected subsets than the accuracy achieved with filter methods (cf. [KJ97, GE03, SIL07]). Nevertheless, the selected subsets are classifier dependent and have a high risk of overfitting (cf. [SIL07]). Like multivariate filter models, wrapper methods model feature dependencies, but are computationally more expensive. Furthermore, deterministic models are prone to getting stuck in a local optimum (greedy search). Compared to deterministic wrapper methods, randomized methods are less prone to getting stuck in a local optimum, but have a higher risk of overfitting (cf. [SIL07]). While some studies (cf. [Str07]) report that non-deterministic wrapper approaches are usually faster than deterministic approaches, the survey in [SIL07] states that randomized algorithms are sometimes slower than deterministic algorithms.

3.2.3 Embedded Approaches

Embedded approaches (cf. [BL97]), sometimes also referred to as *nested subset methods* (cf. [GE03]), act as an integral part of the machine learning algorithm itself. During the operation of the classification process, the algorithm itself decides which attributes to use and which to ignore. Just like wrapper methods, embedded approaches thus depend on a specific learning algorithm, but may be more efficient in several aspects. Since they avoid retraining of the classifier from scratch for every feature subset investigated, they are usually faster than wrapper methods. Moreover, they make better use of the available data by not needing to split the training data into a training and test/validation set.

Decision trees (cf. Section 4.3) are famous examples that use embedded feature selection by selecting the attribute that achieves the "best" split in terms of class distribution at each leave. This procedure is repeated recursively on the feature subsets until some stopping criterion is satisfied. The output is a model that uses only a subset of features – those that appear in the nodes of the decision tree (cf. [Mla06]). For this reason, decision trees are usually not sensitive to the presence of irrelevant features.

Another example of embedded feature selection, which is based on training a non-linear support vector machine (SVM, cf. Section 4.5), was proposed in [Rak03]. In this study, different criteria for variable selection are derived from SVM theory. These criteria are based on (*i*) the weight vector norm and (*ii*) generalization error bounds sensitivity with respect to a variable (see Section 4.5 for details about SVMs). The results show that the criterion based on the weight vector derivative achieves good results. Guyon *et al.* [GWB02] introduced embedded feature selection using the weight vectors of a support vector machine in combination with recursive feature elimination. Their work was applied to the task of gene selection, and results show that their method eliminates gene redundancy automatically and yields subsets that achieve better classification than the full set of features. An approach called *grafting*, which uses fast gradient-based heuristics to find the best feature subset, was introduced in [PLT03]. Iteratively, a gradient-based heuristic is used to quickly assess which feature is most likely to improve the existing model, that feature is then added to the model, and the model is incrementally optimized using gradient descent. Moreover, random forests (see Section 4.6) are also often considered and applied (cf. [DA06]) as embedded feature selection methods, because of their built in possibility to measure descriptor importance as well as the similarity between features.

3.2.4 Comparison of Filter, Wrapper and Embedded Approaches

Wrapper methods usually have the highest computational complexity, followed by embedded approaches. Multivariate filter methods are usually faster than embedded approaches and wrapper methods, but slower than univariate filter methods (feature ranking), which are often a good choice for datasets with a very large numbers of instances. Since wrapper and embedded approaches select the feature subsets based on a specific learning algorithm, they are more prone to overfitting than filter methods which are independent of the later applied classifier. Moreover, deterministic wrapper algorithms are also prone to getting stuck in local optima. On the other hand, wrapper and embedded methods have often proven to achieve very good classification accuracies for specific problems. Due to the wide variety of variables, data distributions, classifiers and objectives, there is no feature selection method that is universally better than others (cf., for example, [GGN06]). A general comparison of filter and wrapper methods as well as embedded approaches is, for example, given in [JC99, LY05b, GGN06, SIL07], where methods are compared in terms of classification performance, general and computational complexity, scalability, model independence, and behavior (are methods prone to getting stuck in local optima or have high risk of overfitting).

3.3 Dimensionality Reduction

In some applications, reducing the dimension of the data is only useful if the original features can be retained (for example, because the interpretability of the original features should not be affected). If these considerations are not of concern, other techniques which reduce the dimension of a dataset can be considered.

Dimensionality reduction (DR) refers to algorithms and techniques which create new attributes as (often linear) combinations of the original attributes in order to reduce the dimensionality of a dataset (cf. [LM98]). Rather than selecting a subset of the features, these techniques involve some type of feature transformation and aim at reducing the dimension such that the representation is as faithful as possible to the original dataset, but with a lower dimension and removed redundancy. Because the new attributes are combinations of the original ones, the transformation process is also referred to as *feature construction* or *feature transformation*. This process of constructing new features can be followed by or combined with a feature subset selection process – the original feature set is first extended by the newly constructed features and then a subset of features is selected (cf. [Mla06]). The study in [Pop01] shows that adding newly computed features to the original attributes can increase the classification results achieved with these feature sets more than replacing the original attributes with the newly computed features.

In contrast to many feature selection methods, dimensionality reduction techniques usually map the data to lower dimension in an *unsupervised* manner, i. e., the class labels are not considered, just the explanatory variables are used.

3.3.1 Low-rank Approximations

Low-rank approximations replace a large and often sparse data matrix with a related matrix of much lower rank. The objectives of these techniques are to reduce the required storage space and/or to achieve a more efficient representation of the relationship between data elements. Low-rank approximations are used within several dimensionality reduction methods, such as principal component analysis (PCA, cf. Section 3.3.2) or singular value decomposition (SVD, cf. Section 3.3.3). Depending on the applied approximation, great care must be taken in terms of storage requirements. If the original data matrix is very sparse (for example, as it is the case for many text mining problems), the storage requirements for the low-rank approximations might be higher than for the original data matrix with much higher dimensions, since the approximation matrices are often almost completely dense. For the factors of the low-rank approximation technique, the sparseness may still be preserved (for example, the SVD factors U, Σ, V, or the NMF factors W and H), but the approximation of the original data matrix is usually almost dense (for example, the product $U\Sigma V^\top$ within SVD, or the product WH within NMF, see later).

Different *matrix factorization* or *matrix decomposition* techniques can be applied for computing low-rank approximations. The most well known techniques are the eigenvalue decomposition and singular value decomposition, but there are several other techniques – some of them are listed at the end of this section. In recent years, an approximation technique for *nonnegative* data – called nonnegative matrix factorization (NMF, cf. Section 3.3.4) – has been used successfully in various fields. In the following, the main characteristics of principal component analysis, singular value decomposition and nonnegative matrix factorization are discussed.

3.3.2 Principal Component Analysis

One of the most widely used DR techniques is the principal component analysis (PCA, cf. [Kra98, Par98, Jol02, Sha03]), which produces new attributes as linear combinations of the original variables. These new attributes – called principle components (PCs) – have to meet the following criteria: The PCs are (*i*) linear combinations of the original attributes, (*ii*) orthogonal to each other, and (*iii*) capture the maximum amount of variation in the data (see paragraph on principal components). Often the variability of the data can be captured by a relatively small number of PCs, and, as a result, PCA can achieve high reduction in dimensionality with usually lower noise than the original patterns. A drawback of PCA is that the PCs are not always easy to interpret, and, in addition to that, PCA depends on the scaling of the data (i. e., removing the mean of each variable). Depending on the field of application, PCA is sometimes also referred to as Karhunen-Loève transform (KLT, cf. [Fuk90]). In the following, the mathematical background of PCA is explained briefly.

Eigenvalues and eigenvectors. For an $n \times n$ matrix C and a nonzero vector $\mathbf{q}$, the values of λ satisfying the equation

$$C\mathbf{q} = \lambda\mathbf{q} \tag{3.6}$$

are called the eigenvalues of C, the vectors $\mathbf{q}$ satisfying Equation (3.6) are called eigenvectors. The number of non-zero eigenvalues of C is at most rank(C), which is the maximal number of linearly independent columns or rows of C, respectively. Equation (3.6) can also be written as $(C - \lambda I_n)\mathbf{q} = 0$.

If C (and hence $(C - \lambda I_n)$) is an $n \times n$ square and full-rank matrix (i. e., C has at most n linearly independent eigenvectors), the eigenvalues of C can be found, for example, by solving the characteristic equation (also called the characteristic polynomial) det(C$-\lambda I_M) = 0$, where $\det(S)$ denotes the determinant of the square matrix S. This method for solving the characteristic equation should be considered as a mathematical proof rather than as a state-of-the-art method. Computational methods are not based on this representation (as it is the case for the algorithms mentioned in the paragraph about computational complexity (see below)).

If C has n linearly independent eigenvectors $\mathbf{q}_1$,...,$\mathbf{q}_n$, then C can be expressed by a product of three matrices

$$C = Q\Lambda Q^{-1}, \tag{3.7}$$

where Λ is a diagonal matrix whose diagonal entries are the eigenvalues of C in decreasing order $(\lambda_1, ..., \lambda_n)$, and $Q = [\mathbf{q}_1, ..., \mathbf{q}_n]$ is the matrix of eigenvectors of C (the i^{th} eigenvector corresponds to the i^{th} largest eigenvalue). Equation (3.7) is called *eigenvalue decomposition* (or *eigendecomposition*) of C. Mathematical details behind the eigendecomposition can be found in the literature, for example, in [GV96] or [Dem97].

Covariance and correlation. The *covariance* of two attributes is a measure of how strongly these attributes vary together. The covariance of a sample of two random variables $\mathbf{x}$ and $\mathbf{y}$ (with mean $\overline{x}$ and $\overline{y}$) with sample size n can be calculated as

$$\mathrm{Cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}). \tag{3.8}$$

If $\mathbf{x} = \mathbf{y}$, then the covariance is equal to the variance. When $\mathbf{x}$ and $\mathbf{y}$ are normalized by their standard deviations σ_x and σ_y, then the covariance of $\mathbf{x}$ and $\mathbf{y}$ is equal to the *correlation coefficient* of $\mathbf{x}$ and $\mathbf{y}$, which indicates the strength *and* the direction of a linear relationship between $\mathbf{x}$ and $\mathbf{y}$. The correlation coefficient can be calculated as

$$\mathrm{Corr}(\mathbf{x}, \mathbf{y}) = \frac{\mathrm{Cov}(\mathbf{x}, \mathbf{y})}{\sigma_x \sigma_y}. \tag{3.9}$$

Given an $m \times n$ matrix A, whose m rows are data objects and whose n columns are attributes, the covariance matrix Cov(A), which is a square matrix constructed of the single covariances, can be calculated. If the values of each attribute of A are shifted such that the mean of each attribute is 0, then Cov(A)= $A^\top A$.

Principal components. If the eigenvalue decomposition is performed on the square matrix Cov(A), then the original data matrix A can be transformed to another matrix $A' := AQ$, with $Q = [\mathbf{q}_1, ..., \mathbf{q}_n]$ (see also Equation (3.7)). Each column of A' is a linear combination of the original attributes, the columns of A' are the principal components, the variance of the i^{th} new attribute is λ_i, and the sum of the variances of all new attributes is equal to the sum of the variances of the original attributes.

The eigenvalue decomposition can also be performed on Corr(A). Chapter 6 provides some detailed discussions and comparisons about the application of the eigenvalue decomposition using the covariance matrix or the correlation coefficient matrix. In both cases (i. e., using Cov(A) or Corr(A)), A' satisfies the following properties (cf. [TSK05]):

i Each pair of the new attributes has covariance 0,

ii The new attributes are ordered descendingly with respect to their variance,

iii The first new attribute (i. e., the first principal component) captures as much of the variance of the data as possible by a single attribute, and,

iv Each successive attribute captures as much of the remaining data as possible.

The eigenvectors of Cov(A) (or Corr(A), respectively) define a new set of orthogonal axes, and the eigenvector associated with the largest eigenvalue indicates the direction in which the data has the most variance. PCA can be viewed as a rotation of the original axes, where the total variability of the data is still preserved, but the new attributes are now uncorrelated.

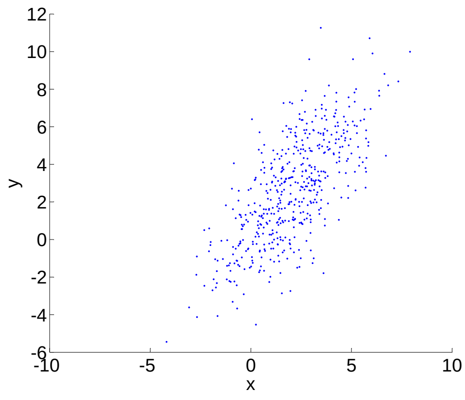

Figure 3.3: Data before the PCA transformation.

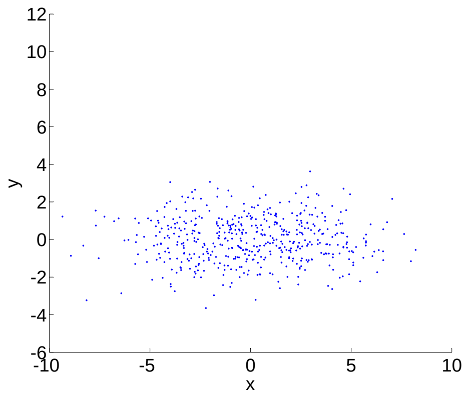

Figure 3.4: Data after the PCA transformation.

Figures 3.3 and 3.4 show the alignment of the new axes in the directions of the maximum variability of the data. The figures show a plot of two dimensional data points, before and after a PCA transformation. The total variance of the original data points (shown in Figure 3.3) is equal to the total variance of the new attributes, but the first new attribute captures as much of the variance of the data as possible. In Figure 3.3, the variance on the x-axis is 3.84, the variance on the y-axis is 7.99, and the total variance is thus 11.83. After the transformation (Figure 3.4), the total variance remains unchanged, but the variance on the x-axis is increased to 10.36 and the variance on the y-axis is reduced to 1.47.

Advantages and drawbacks. PCA is a relatively simple, non-parametric, generic method that is useful for finding new, more informative, uncorrelated features and it can be used to reduce dimensionality by rejecting low variance features. Since the principal components are orthogonal to each other, every principal component is uncorrelated to every other principal component (i. e., they do not contain any redundant information). The principal components are designed to account for the highest percentage of the variation among the variables with as few PCs as possible. Thus, often the first few PCs account for some large percentage of the total variance, allowing for a compact representation of the data with only low dimensions.

However, PCA is limited to re-expressing the data as combinations of its basis vectors. A main drawback of PCA is that each PC is a linear combination of *all* original variables, thus leading to a potentially difficult interpretation of the PCs. On the contrary, in a system with many variables PCA may be used to project the dimension down to a reasonable number of plots, and the principal components could be rotated towards a more meaningful representation. Moreover, PCA is sensitive with respect to the units of measurement. If the units and the variances of attributes vary a lot, then variables with high variance tend to dominate the first few principal components. In this case, the data need to be normalized prior to the PCA transformation.

Computational complexity. From a computational point of view, the eigenvalue decomposition for solving the PCA transformation is rather expensive in terms of runtime, especially for a large number of attributes. There are several algorithms for solving symmetric eigenproblems, but all of them are of order $O(n^3)$. Demmel [Dem97] discusses several algorithms for computing the eigenvalue decomposition, such as tridiagonal QR iteration (see also [GV96]), bisection and inverse iteration, the divide-and-conquer algorithm (see also [Cup81]), Rayleigh quotient iteration (see also [Par98]), as well as the cyclic Jacobi method (see also [GV96]). The first three algorithms are reported to be the fastest (cf. [Dem97]) – however, they are still of order $O(n^3)$. A recent comparison of the performance of `LAPACK` [1] implementations (a software package that offers fast linear algebra routines) of these three algorithms can be found in [DMP08].

3.3.3 Singular Value Decomposition

Equation (3.7) decomposes a *square* matrix into a product of three matrices. An arbitrary matrix can be decomposed in a similar way (although eigenvectors do not exist for non-square matrices). The singular value decomposition (SVD, cf. [Dem97, BDJ99]), is a more general method that factors *any* $m \times n$ matrix A of rank r into a product of three matrices, such that

$$A = U\Sigma V^\top, \tag{3.10}$$

where U is $m \times m$ (the left singular vectors), Σ is $m \times n$, and is V is $n \times n$ (the right singular vectors). U and V are both orthonormal matrices (i. e., $UU^\top = I_m$ and $VV^\top = I_n$), and Σ is an $m \times n$ matrix where the nonnegative and descendingly ordered elements along the diagonal of the left/top quadratic subblock are the singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_r \geq 0$ of A. All other elements of Σ are 0.

If $r < min(m, n)$, i. e., the matrix A is not full-rank, then only r singular values are greater than 0. A full-rank decomposition of A is usually denoted like this: $A_r = U_r\Sigma_r V_r^\top$. The singular values are always real numbers. If the matrix A is real, then U and V are also real. The relative sizes of the matrices U, Σ and V are shown in Figure 3.5. All entries not explicitly indicated in Σ are zero.

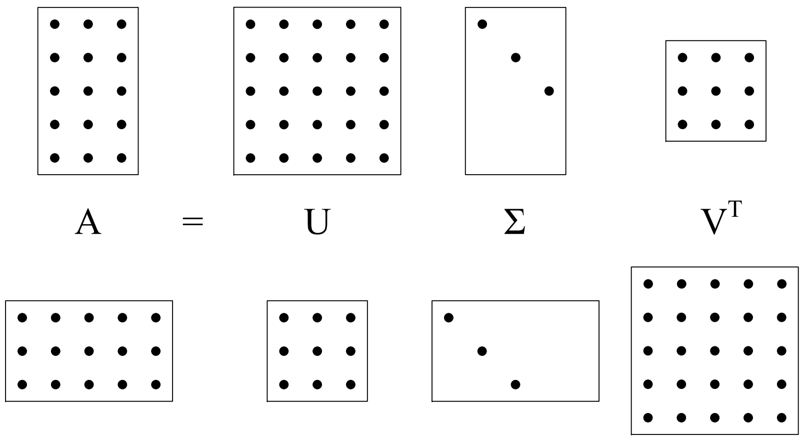

Figure 3.5: Illustration of the SVD (following [BDJ99]). For $m > n$ (upper part of the figure), and $m < n$ (lower part).

The *truncated* SVD (or reduced rank SVD) to A can be found by setting all but

[1] `http://www.netlib.org/lapack`

the first k largest singular values equal to zero and using only the first k columns of U and V. This is usually denoted like

$$A_k = U_k \Sigma_k V_k^\top, \tag{3.11}$$

or, more explicitly

$$A_k \simeq (\mathbf{u}_1, \dots, \mathbf{u}_k) \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_k^\top \end{pmatrix}, \tag{3.12}$$

where $(\mathbf{u}_1, \dots, \mathbf{u}_k)$ are the column vectors of U, and $(\mathbf{v}_1^\top, \dots, \mathbf{v}_k^\top)$ are row vectors of $V^\top$.

A theorem by Eckart and Young from 1936 [EY36] states that amongst all possible rank k approximations, A_k is the best approximation in the sense that $||A - A_k||_F$ is as small as possible (cf. [BDJ99]), in other words, the singular value decomposition gives the closest rank k approximation of a matrix, such that

$$||A - A_k||_F \leq ||A - B_k||_F, \tag{3.13}$$

where B_k is *any* matrix of rank k, and $||.||_F$ is the Frobenius norm, which is defined as $(\sum |a_{ij}|^2)^{1/2} = ||A||_F$) (cf. [Dem97]). Since both, premultiplying an $m \times n$ matrix X with an $m \times m$ orthogonal matrix, and postmultiplying X with an $n \times n$ orthogonal matrix leaves the Frobenius norm unchanged, it can be shown that the Frobenius norm of A is equal to the Euclidean norm of the vector containing all singular values

$$||A||_F = ||U\Sigma V^\top||_F = ||\Sigma V^\top||_F = ||\Sigma||_F = \sqrt{\sum_{j=1}^{r_A} \sigma_j^2}. \tag{3.14}$$

Computational complexity. In general, the computational complexity of the original Golub-Reinsch SVD algorithm for computing all three matrices U, Σ and V is $O(4m^2n + 8mn^2 + 9n^3)$, for computing only the matrices Σ and V it is $O(4mn^2 + 8n^3)$ (cf. [GV96]), making the decomposition unfeasible for large datasets. The R-SVD algorithm introduced by Chan [Cha92] has a complexity of $O(4m^2n + 22n^3)$ for computing all three matrices, and a complexity of $O(2mn^2 + 11n^3)$ for computing only Σ and V. The Lanczos method (cf. [GV96, Par98]) is able to compute a thin SVD (i. e., r – the number of required singular values – must be known in advance) in $O(mnr^2)$, but is known to be inaccurate for smaller singular values (cf. [Ber92]). [Bra02] proposed an SVD update algorithm for dense matrices, which shows a linear time complexity in the size and the rank of the data, $O(mnr)$.

Relation to PCA. Calculating the SVD is equivalent of finding the eigenvalues and eigenvectors of $AA^\top$ and $A^\top A$. The eigenvectors of $AA^\top$ make up the columns of U, the eigenvectors of $A^\top A$ make up the columns of V. Moreover, the singular values in Σ are the square roots of eigenvalues from $AA^\top$ and $A^\top A$. The SVD-PCA connection stems from the straightforward linear algebra calculations

$$AA^\top = (U\Sigma V^\top)(V\Sigma^\top U^\top) = U\Sigma^2 U^\top, \text{and} \tag{3.15}$$

$$A^\top A = (V\Sigma^\top U^\top)(U\Sigma V^\top) = V\Sigma^2 V^\top. \tag{3.16}$$

A comparison with Equation (3.7) shows the following: As already mentioned before, if the values of each attribute of A are shifted such that the mean of each attribute is 0, then Cov(A)=$A^\top A$. If we re-write Equation (3.7) in the form $A^\top A = Q\Lambda Q^{-1}$, and replace the left part of Equation (3.16) with the right part of Equation (3.7), we get $Q\Lambda Q^{-1} = V\Sigma^2 V^\top$. Since Q and V are orthonormal, $Q^{-1} = Q^\top$, and $V^{-1} = V^\top$. If the columns of Λ and Q are now arranged such that the eigenvalues in Λ are descendingly ordered, then it can be seen that Λ is identical to the square of Σ, and Q is identical to V.

An often mentioned difference between these two approaches is that PCA removes the mean of each variable whereas SVD uses the original data (however, the mean could also be removed before computing the SVD). Especially for sparse data it is not always preferable to remove the mean of the data (cf. [TSK05]). The similarities and dissimilarities of PCA and SVD have been examined in various studies, for example Cherry [Che97] provides an excellent comparison of these two techniques.

Software. There are several software libraries for numerical linear algebra implementations (`LAPACK` [2], `ScaLAPACK` [3], `SVDPACK` [4], . . .) that contain various algorithmic variants for computing the SVD (and also the eigenvalue decomposition for PCA).

3.3.4 Nonnegative Matrix Factorization

The nonnegative matrix factorization (NMF, cf. [PT94, LS99]) consists of reduced rank *nonnegative* factors $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$ with (problem dependent) $k \ll min\{m,n\}$ that approximate a given nonnegative data matrix $A \in \mathbb{R}^{m\times n}$: $A \approx WH$. The non-negativity constraints require that *all entries* in A, W and H are zero or positive. Despite the fact that the product WH is only an approximate factorization

[2]`http://www.netlib.org/lapack`
[3]`http://www.netlib.org/scalapack`
[4]`http://www.netlib.org/svdpack`

of A of rank at most k, WH is called a nonnegative matrix factorization of A. The non-linear optimization problem underlying NMF can generally be stated as

$$\min_{W,H} f(W,H) = \frac{1}{2}||A - WH||_F^2, \tag{3.17}$$

where $||.||_F$ is the Frobenius norm (see Section 3.3.3). Although the Frobenius norm is commonly used to measure the error between the original data A and WH, other measures are also possible, for example, an extension of the Kullback-Leiber divergence to positive matrices (cf. [DS05]), a convergence criterion based on the the Karush-Kuhn-Tucker (KKT) condition (cf. [KP08]), or an angular measure based on the angle θ_i between successive basis vectors, i. e. $W_i^{(j+1)}$ and $W_i^{(j)}$ (cf. [LMA06]). Unlike the SVD, the NMF is not unique, and convergence is not guaranteed for all NMF algorithms. If they converge, then they usually converge to local minima only (potentially different ones for different algorithms). Fortunately, the data compression achieved with only local minima has been shown to be of desirable quality for many data mining applications (cf. [LMA06]).

Due to its nonnegativity constraints, NMF produces so-called "additive parts-based" (or "sum-of-parts") representations of the data (in contrast to many other linear representations such as SVD, PCA or ICA). This is an impressive benefit of NMF, since it makes the interpretation of the NMF factors much easier than for factors containing positive and negative entries, and enables NMF a non-subtractive combination of parts to form a whole [LS99]. For example, the features in W (called "basis vectors") may be topics of clusters in textual data, or parts of faces in image data. Another favorable consequence of the nonnegativity constraints is that both factors W and H are often naturally sparse (see, for example, the update steps of Algorithm 3, where negative elements are set to zero).

3.3.5 Algorithms for Computing NMF

NMF algorithms can be divided into three general classes: multiplicative update (MU), alternating least squares (ALS) and gradient descent (GD) algorithms. A review of these three classes can be found, for example, in [BBL07]. Pseudo code for the general structure of all NMF algorithms is given in Algorithm 1.

In the basic form of most NMF algorithms, the factors W and H are initialized randomly. Since the approximation quality and convergence of NMF algorithms strongly depends on the initialization of the factors, NMF algorithms are usually repeated several times, each time with newly initialized factors. The variable *maxrepetition* in line 2 specifies the number of repetitions of the complete algorithm. Most algorithms need pre-initialized factors W and H, but some algorithms (for example,

Algorithm 1 – General structure of NMF algorithms.

```
1: given matrix $A \in \mathbb{R}^{m \times n}$ and $k \ll min\{m, n\}$:
2: for $rep = 1$ to $maxrepetition$ do
3:     $W = \text{rand}(m, k)$;
4:     $(H = \text{rand}(k, n);)$
5:     for $i = 1$ to $maxiter$ do
6:         perform algorithm specific NMF update steps
7:         check termination criterion
8:     end for
9: end for
```

the ALS algorithm) only need one pre-initialized factor (as indicated with brackets in line 4 of Algorithm 1). In each repetition, NMF update steps are processed iteratively until a maximum number of iterations is reached (*maxiter*, line 5). Different update steps for the three basic NMF algorithms are briefly described in the following paragraph. If the approximation error of the algorithm drops below a pre-defined threshold, or the shift between two iterations is very small, the algorithm might stop before all iterations are processed (for details, see paragraph *termination criteria*).

Multiplicative update algorithm. The update steps for the multiplicative update algorithm given in [LS01] are based on the mean squared error objective function. Adding ε in each iteration avoids division by zero. A typical value used in practice is $\varepsilon = 10^{-9}$.

Algorithm 2 – Update steps for the multiplicative update algorithm.

```
1: $H = H \,.\!*\, (W^\top A) \,./ (W^\top W H + \varepsilon)$;
2: $W = W \,.\!*\, (A H^\top) \,./ (W H H^\top + \varepsilon)$;
```

The operator ./ divides each entry of the matrix on the left hand-side of the operator by the corresponding entry of the matrix on the right hand-side of the operator. Similarly, the operator $.*$ indicates an element-by-element multiplication.

Alternating least squares algorithm. Alternating least squares algorithms were first mentioned in [PT94]. In an alternating manner, a least squares step is followed by another least squares step. As can be seen in Algorithm 3, all negative elements resulting from the least squares computation are set to 0 to ensure nonnegativity. The standard ALS algorithm only needs to initialize the factor W, the factor H is computed in the first iteration. This algorithm also works with a pre-initialized factor H (lines 3 and 4 in Algorithm 3 have to be executed prior to lines 1 and 2).

Gradient descent algorithm. This third algorithm always takes a step in the

Algorithm 3 – Update steps for the alternating least squares algorithm.

```
1: solve for $H : W^\top WH = W^\top A$;
2: set all negative elements in $H$ to 0;
3: solve for $W : HH^\top W^\top = HA^\top$;
4: set all negative elements in $W$ to 0;
```

direction of the negative gradient, the direction of steepest descent (which can be computed using the partial derivatives for H and W, respectively). Here, ϵ_H

Algorithm 4 – Update steps for the gradient descent algorithm.

```
1: $H = H - \epsilon_H \nabla_H f(W, H)$
2: $W = W - \epsilon_W \nabla_W f(W, H)$
```

and ϵ_W are step-size parameters which have to be chosen carefully in order to get a good approximation (cf. the discussion in [Lin07]). The partial derivatives are $\nabla_H f(W, H) = W^\top(WH - V)$ and $\nabla_W f(W, H) = (WH - V)H^\top$, respectively.

An interesting study investigating proposing and discussing an adaption of gradient descent algorithm was published by Lin [Lin07]. In this paper, the author proposed the use of a projected gradient bound-constrained optimization method for computing the NMF. Results show that this method is is computationally very competitive and appears to have better convergence properties than the standard MU approach in some cases.

All three algorithms are iterative and their convergence depends on the initialization of W (and H). Since the iterates generally converge to a local minimum, often several instances of the algorithm are run using different random initializations, and the best of the solutions is chosen. A proper non-random initialization of W and/or H (depending on the algorithm) can avoid the need to repeat several factorizations. Moreover, it may speed up convergence of a single factorization and reduce the error as defined in Equation (3.17). In Chapter 8 we introduce, discuss and compare new initialization methods for NMF.

Termination criteria. The simplest possible convergence criterion, which is used in almost all NMF algorithms, is to run for a fixed number of iterations (denoted *maxiter* in Algorithm 1). Since the most appropriate value for *maxiter* is problem-dependent, this is not a mathematically appealing way to control the number of iterations, but applies when the required approximation accuracy does not drop below a pre-defined threshold ϵ, even after several iterations. The quality of the approximation accuracy is another convergence criterion. Obviously, the choice of ϵ depends on the size and structure of the data. As already mentioned in Section 3.3.4, different convergence measures can be applied, such as the Frobenius norm (see Equation (3.17)), Kullback-Leiber divergence, KKT, or angular measures. An third convergence criterion is based on the relative change of the factors W and H from one iteration to another. If this change is below a pre-defined threshold δ, then the algorithm also terminates.

Computational complexity of NMF algorithms A single update step of the MU algorithm has the complexity $O(kmn)$ (since A is $m \times n$, W is $m \times k$ and H is $k \times n$), see, for example, [LCL07, RM09]. Considering the number of iterations i of the NMF yields an overall complexity of $O(ikmn)$. The computational complexity of basic GD methods depends on computing the partial derivatives in Algorithm 4, which yields also a complexity of $O(ikmn)$, and is also based on the effort needed for retrieving the optimal size of the step-sizes parameters (cf. [Lin07]). For the ALS algorithm, the complexity for solving the equations in lines 1 and 3 of Algorithm 3 need to be considered additionally. In its most general form, these equations are solved using an orthogonal-triangular factorization.

3.3.6 Other Dimensionality Reduction Techniques

The goal of a **factor analysis** (FA, cf. [Gor83, JW07]) is to express the original attributes as linear combinations of a small number of hidden or latent attributes. The factor analysis searches for underlying (i. e., hidden or latent) features that summarize a group of highly correlated attributes. The following equation shows the relationship between the old and new data objects for a standard factor analysis model (assuming that the mean of each attribute is zero):

$$\mathbf{a}_{i*}^\top = \Lambda \mathbf{f}_{i*}^\top + \epsilon, \tag{3.18}$$

where $\mathbf{a}_{i*}$ is the i^{th} row vector of the original $m \times n$ data matrix A (each row of A corresponds to an instance), and $\mathbf{f}_{i*}$ is the corresponding row vector of the new $m \times p$ data matrix F. Λ is an $m \times p$ matrix of factor loadings, that indicate how much each single original attribute depends on the new attributes (also called *latent* or *common factors*). ϵ is an error term that accounts for the portion of the attributes not accounted by F. Several mathematical and statistical methods are needed to extract the factors, including principal component analysis, maximum-likelihood, and least-squares (cf., for example, [Att99]).

Independent components analysis (ICA, cf. [HKO01]) – a member of the class of blind source separation (BSS) methods – is a method for finding underlying factors or components from multivariate (multidimensional) statistical data. More precisely, ICA separates multivariate signals into additive subsets, that are both statistically independent and non-gaussian.

Multidimensional scaling (MDS, cf. [CC01]) finds projections of the data to a lower dimensional space, while preserving pairwise distances (based on an objective function). Usually, MDS explores similarities and dissimilarities in the data. *FastMap* [FL95], a fast incremental technique, and *ISOMAP* [TSL00], a method used to handle data with non-linear relationship to one another, are two famous examples within the set of MDS techniques.

Locally linear embedding (LLE, cf. [RS00]) is an unsupervised, non-linear and non-parametric dimensionality reduction technique that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional input. LLE maps data into a single global coordinate system of usually much lower dimensionality, without being prone to getting stuck in local minima.

A classical method for the lossy compression of datasets (instances, not attributes) is **vector quantization** (cf. [LBG80]). This method models the probability density functions by the distribution of *prototype vectors* (also called *code vectors*). A vector quantizer divides a large set of k-dimensional vectors into a finite set of prototype vectors $Y = y_i : i = 1, 2, ..., N$, where each y_i acts as a centroid point (comparable to centroids in clustering algorithms), and all y_i have approximately the same number of points closest to them. Vector quantization has been used for image and signal compression (cf. [GG91]), voice recognition (cf. [KL93]), and for general statistical pattern recognition (cf. [SDM06]).

3.3.7 Comparison of Techniques

When comparing dimensionality reduction techniques, various key issues of the methods have to be taken into account. Important aspects are quality and type of representation, theoretical provability, determinism, computational complexity, availability of algorithms, etc. An important property of DR methods is the aspect if the representation captures the characteristics of the data that are important while eliminating aspects that are irrelevant or even detrimental (e. g., noise). The answer to this question depends to a large extend on the characteristic of the data. Some DR methods capture linear relationships between the old and the new sets of attributes (PCA, SVD, FA), while other methods such as ISOMAP and LLE are also able to capture with nonlinear relationships. Due to the non-negativity constraints of NMF, this technique is able to produce "sum-of-parts" representations of the original data (cf. Section 3.3.4), which makes the interpretation of the NMF factors much easier than for other methods.

From a theoretical point of view, PCA and SVD have the best theoretically provable properties, and give the best approximation in terms of approximation error of all methods (cf. Equation (3.13) in Section 3.3.3). Moreover, both techniques are purely deterministic and always produce reproducible results (for identical input). NMF algorithms based on non-randomly initialized factors are also deterministic, while randomly initialized factors produce different answers on different runs. Factor analysis and MDS techniques are also non-deterministic. The computational complexity of most methods is over $O(m^2)$ (cf. [TSK05]), where m is the number of objects. Only FastMap has a linear time and space complexity.

The availability of fast and efficient algorithms is also an essential aspect of dimensionality reduction methods. The software libraries mentioned in Section 3.3.3 provide several algorithmic variants for efficiently computing singular value decomposition and eigenvalue decomposition. Moreover, SVD and eigenvalue decomposition algorithms are integrated into Matlab[5], and methods for computing NMF and factor analysis are included into the Matlab Statistics Toolbox.

In Part II of this thesis, we focus on three of these methods (PCA, SVD, NMF) for investigating the relationship between dimensionality reduction methods and the resulting classification accuracy in the context of email filtering and drug discovery applications. Summarizing, SVD, PCA and NMF contain several crucial properties that make them preferable over other methods in the context of this thesis. PCA and SVD benefit from their theoretical provability and the fact, that they are able to produce the best approximation of the original data compared to any other method. Contrary to that, NMF seems to be a very promising method since the "sum-of-parts" representation obtained from the non-negativity constraints of NMF provides the possibility to interpret the factors ("basis vectors" or "basis features").

[5]`http://www.mathworks.com/`

Chapter 4

Supervised Learning

In Chapter 2 of this thesis, machine learning algorithms were classified into two distinct groups – supervised (*predictive*) and unsupervised (*descriptive*) learning methods. Algorithms for supervised learning use a set of labeled training examples, each with a feature vector and a class label. Given some training data, most algorithms produce a classifier model that maps an object to a class label. Contrary to that, unsupervised learning algorithms aim at finding similarities among unlabeled objects and work without the class information of a sample (cf. Section 2.4.4). In this thesis, the focus is on the application of supervised learning algorithms to the task of classification in order to compare and evaluate the classification performance of these algorithms based on several subsets of features extracted by feature selection and dimensionality reduction techniques discussed in Chapter 3.

In this chapter, all classification models used in Part II of this thesis are explained and discussed separately. Moreover, the algorithms are compared to each other in terms of classification accuracy, comprehensibility, complexity and speed. The discussed methods comprise k-nearest neighbor classifiers, decision trees, rule-based learners, support vector machines and several ensemble methods (including bagging and random forest). Moreover, two techniques originating from web search and text mining – the vector space model and its extension, latent semantic indexing – are discussed. At the end of this chapter, some classification methods which are not used in Part II of this thesis, but which have been used in the contexts of the application areas discussed in Chapter 5 are summarized briefly. Relevant literature for each learning algorithm is given directly in the appropriate sections discussing these algorithms.

4.1 Relevant Literature

Similarly to Chapter 6, relevant literature for supervised learning algorithms is again included directly in the main text.

4.2 The k-Nearest Neighbor Algorithm

The k-nearest neighbor (kNN) algorithm (cf. [CH67, Aha92, CH95]) is a simple and one of the most intuitive machine learning algorithms that belongs to the category of *instance-based learners.* Instance-based learners are also called *lazy learner* because the actual generalization process is delayed until classification is performed, i. e., there is no model building process. Unlike most other classification algorithms, instance-based learners do not abstract any information from the training data during the learning (or training) phase. Learning (training) is merely a question of encapsulating the training data, the process of generalization beyond the training data is postponed until the classification process.

kNN is based on the principle that instances within a dataset will generally exist in close proximity to other instances that have similar properties (cf. [CH67]). If the objects are tagged with a classification label, objects are classified by a majority vote of their neighbors and are assigned to the class most common amongst its k-nearest neighbors. k is a usually rather small odd (to avoid tied votes) positive number and the correct classification of the neighbors is known a priori. The objects can be considered n-dimensional points within an n-dimensional instance space where each point corresponds to one of the n features describing the objects. The distance or "closeness" to the neighbors of an unclassified object is determined by using a *distance metric* (also called *similarity function*), for example the Euclidean distance or the Manhattan distance. A survey of different distance metrics for kNN classification can be found, for example, in [WBS06].

The high degree of local sensitivity makes kNN highly susceptible to noise in the training data – thus, the value of k strongly influences the performance of the kNN algorithm. The optimal choice of k is a problem dependent issue, but techniques like cross-validation (cf. Section 2.4.4) can be used to reveal the optimal value of k for objects within the training set. Figure 4.1 is an example of how the choice of k affects the outcome of the kNN algorithm. In Figure 4.1 (a), if k is set to 7 or smaller, a majority count of the k-nearest neighbors of the object under investigation (q_i) returns the (obviously correct) class *crosses.* Increasing k to 9 (all objects inside the big circle) would result in predicting q_i to class *circles.* On the contrary, setting k to 1 or 9 in Figure 4.1 (b) returns class *circles*, if k is set to 3, 5, or 7, the kNN

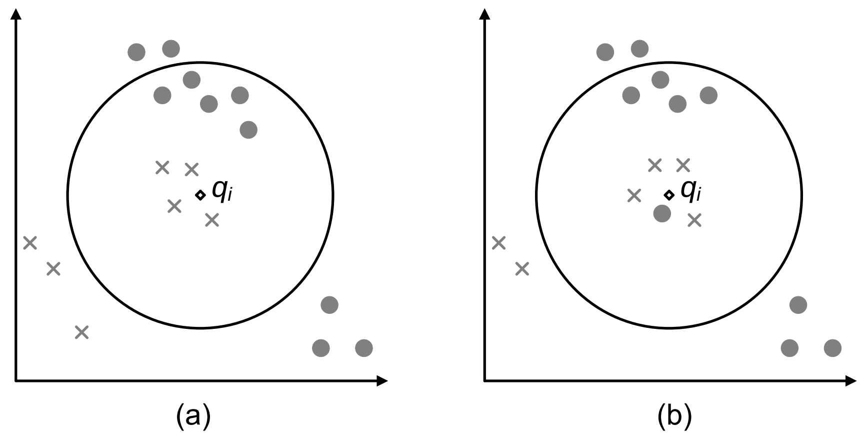

Figure 4.1: kNN classifier.

algorithm returns class *crosses*.

Since the procedure of finding the closest object(s) of a training set is linear in the number of training instances for every unlabeled object, the time to make a single prediction is proportional to the number of training instances. Moreover, every additional feature further increases the classification time. As kNN has absolutely no built-in process of selecting important features, feature reduction is particularly important for this algorithm to scale down the required runtime. Especially feature ranking (cf. Section 3.2) is often used to discard features with weights below a certain threshold value and to improve classification accuracy of the classifier (cf., for example, [Aha92]).

General evaluation. Considering the simplicity of the kNN algorithm, the classification results of kNN are generally quite good and comparable to the performance achieved with decision trees and rule-based learners (cf. [Kot07]). However, the classification accuracy of kNN models does in general not reach the accuracy achieved with support vector machines or ensemble learners. kNN is considered to be intolerant to noise, since its similarity measures can easily be distorted by errors in the attribute values, and is also very sensitive to irrelevant features. On the contrary, kNN models are usually not prone to overfitting and can be applied to incremental learning strategies – since kNN does not build a classification model, newly classified instances can be added to the training set easily.

Relevant literature. There are several studies that survey the application of kNN for classification tasks. Besides almost all introductory data mining books and surveys mentioned in Section 2.5, there are, for example, the book by Mitchell [Mit98] or a recent survey article by Jiang *et al.* [JCW07], that summarizes several improvements of kNN algorithms for classification. An interesting publication on weighted kNN classification based on symbolic features was published by Cost and

Salzberg [CS93]. Distance tables are calculated to produce real-valued distances from features coming from symbolic domains. The authors state that their technique is comparable or superior to standard kNN in three different application domains and has advantages in training speed and simplicity. A different study, published by Han *et al.* [HKK01], focuses on a weight-adjusted kNN implementation which finds the optimal weight vector using an optimization function based on the leave-out-out cross-validation and a greedy hill climbing technique. Moreover, two performance optimizations based on intelligent selection of words followed by a document clustering process, are introduced. The results show that the classification accuracy remains generally unchanged while the computational performance could be improved by a few orders of magnitude. Gracia *et al.* [GDB08] investigated high-performance aspects of kNN classification based on utilizing GPU performance and show that kNN can be accelerated up to a factor of 120 when using specific GPU settings.

4.3 Decision Trees

Decision trees (cf. [Kas80, BFS84, Qui93, SL98], sometimes also called *classification trees* or *regression trees*) apply a "divide-and-conquer" approach to the problem of learning from a set of independent instances. All decision trees are structured such that the root node and the internal nodes contain attribute test conditions to separate (classify) records that have different characteristics. The branches between nodes represent conjunctions of features that lead to those classifications, and leaf nodes represent the class labels.

Since the problem of constructing optimal (binary) decision trees is an NP-complete problem (cf., for example, [Qui93]), heuristics are used for constructing near-optimal trees. The feature that best divides the training data is used to partition the records into smaller subsets. The crucial step is how to determine which feature to split on. Different *feature evaluation rules* are used to search for the best splitting feature. Some of these rules are derived from *information theory*, most of them being based on Shannon's entropy (cf. [Sha51]). Information gain (cf. Section 3.2.1), a technique which can also be applied as feature ranking method, is one of the most widely used splitting criteria. Another information theory based splitting criterion is to maximize the global mutual information by expanding tree nodes that contribute to the largest gain in average mutual information of the whole tree (cf., for example, [LWL03]). Rules derived from *distance measures* between groups calculate the separability, divergence or discrimination between classes. Popular distance measures are the Gini index of diversity (cf. [BFS84]), a measure of the inequality of a distribution, the Kolmogorov-Smirnov distance (cf. [Sha03]), a measure of the underlying similarity of two samples, or the χ^2-statistic (cf. [LS96]), a statistical significance test that uses the χ^2-distribution to test the fit between a theoretical frequency distribution and a frequency distribution of observed data for which each observation may fall into one of several classes The χ^2-distribution is the sum of the squares of a set of variables or attributes, each of which has a normal distribution and is expressed in standardized unit.

Basic algorithm. The typical decision tree algorithm proceeds as follows: First, select an attribute to place at the root node and make one branch for each possible value (or interval) of this attribute. This splits up the training set into subsets, one for every value (interval) of the attribute. Then, repeat the process recursively for each branch. If at any time all instances at a node have the same classification, stop developing that part of the tree. Algorithm 5 presents a pseudo-code for building decision trees.

Algorithm 5 – Pseudo-code for building decision trees.

```
Given a set set of training records $D(t)$ that are associated with node $t$
and class labels $y = \{y_1, y_2, \ldots, y_c\}$
.........................................................................
Apply recursively for each child node
if all records in recordset $D(t)$ belong to the same class $y_t$ then
  $t$ is a leaf node labeled as $y_t$
else
  $D(t)$ contains records that belong to more than one class
  Select evaluation rule to partition the records into smaller subsets
  Create a child node for each outcome of test condition
  Distribute the records in $D(t)$ to the children
end if
```

Three major groups of decision tree algorithms are used in practice. The most well-known algorithms are C4.5 and its successor C5.0, which are extensions of the ID3 algorithm, all introduced by Quinlan [Qui93]. ID3, C4.5 and C5.0 trees use information gain as splitting criterion and can handle both, categorical and numeric attributes (cf. Section 2.3). Moreover, C4.5 is able to deal with missing values, i. e., objects, where one or more attribute values are missing. The commercial successor C5.0 is faster and more memory efficient than C4.5, and supports boosting (cf. Section 4.6), weighting the attributes and winnowing the data (i. e., discarding those attributes that appear to be only marginally relevant). The CART (*classification and regression trees*) algorithm developed by Breiman [BFS84] uses the Gini index (see last paragraph) as evaluation rule and allows only for binary splits (in contrast to

C4.5). The CHAID (*chi-square automatic interaction detectors*) algorithm [Kas80] uses the χ^2 test (see last paragraph) as splitting function and a statistical stopping rule that discontinuous tree growth. Like C4.5, this algorithm allows for multiple splits.

Since decision trees are often prone to overfitting (cf. [WF05], high generalization error due to too many attributes, cf. Section 2.4.4), the trees are usually pruned, i. e., the number of tests along a certain path is reduced. The tree is either not grown to its full size, or subtrees of the fully grown tree are replaced with leafs subsequent to the model building process. Pruned trees are also more easily comprehensible than full trees, and the classification process can be performed faster since the number of attributes under investigation is reduced.

General evaluation. The comprehensibility of decision trees is one their most useful characteristics, since domain experts can easily understand the principle of the tree, and why a certain object is classified to belong to a specific class. Moreover, decision trees are probably the most extensively researched machine learning method, can deal with any kind of input data (discrete, continuous, binary,... attributes). They can also cope with missing values (see Section 2.4.2), since the information that attribute values are missing for specific objects can be processed by most decision tree algorithms. The learning process of decision trees is usually quite fast compared to other methods like support vector machines or neural networks, and since most trees are pruned, their classification process is usually also very fast. Several studies (cf. [WF05, Kot07, JGD08]) have shown that the classification accuracy is generally comparable to the quality of kNN and rule-based learners, but cannot reach the quality of support vector machines or embedded methods, which, on the contrary, are hardly comprehensible (difficult to understand for domain experts) and are not good in handling missing data (since missing data has to be replaced with alternative values such as mean or zero values before classification).

Relevant literature. Besides the introductory work on the three major decision tree algorithms mentioned before (cf. [Kas80, BFS84, Qui93]), there are several studies that provide comprehensive surveys on the application of various decision tree algorithms in the fields of data mining and machine learning, for example, [Mur98] or [SL98]. Breslow and Aha [BA97] reviewed methods of tree simplification to improve the comprehensibility of the trees. Recently, several publications focused on the parallel and distributed potential of decision tree algorithms. A study by Yildiz and Dikmen [YD07] analyzed the parallelization potential of C4.5 algorithms, and provided three ways to parallelize them. The algorithms are parallelized by distributing either the features, the data or, the nodes among several slave processors. Experimental results show that the node based parallelization implementation demonstrates the best speedup among these three approaches. An article by Bhaduri *et al.* [BWG08] focused on a distributed algorithm for decision tree induction in large peer-to-peer environments. The authors state, that with sufficient time, the algorithm converges to the same tree given all the data of all the peers, and mention that this algorithm is suitable for scenarios in which the data is distributed across large P2P networks (since the algorithm seamlessly handles data changes and peer failures). Other recent work on the distribution of decision trees includes the induction of multi-class and multi-feature split decision trees from distributed data [OPS09]. This study describes a method that generates compact trees using multi-feature splits in place of single-feature splits and is based on Fisher's linear discriminant function, which aims at finding the projection vector such that the projection of the original data has the best discriminantability (cf. [DHS01]). Moreover, a new scheme on a privacy-preserving distributed decision tree algorithm was introduced in [FYS09]. Their work presents a decision-tree mining algorithm based on homomorphic encryption technology.

4.4 Rule-Based Learners

The output of decision trees can easily be mapped into a set of rules. Each path from the root node to the leafs in the trees can be considered as a separate rule (one rule for each leaf). Rule-based algorithms (cf., for example, the overview of Fuernkranz [Fue99]) can create rules directly from the training data without creating a tree. Most rule-based learners classify records using a collection of *if ... then* rules. The rules are represented in a disjunctive normal form (DNF), where a logical formula is represented as one or more disjunctions of one or more conjunctions of one or more literals, such as $S_R = (r_1 \vee r_2 \vee \cdots \vee r_k)$, where S_R is the *rule set*, and r_i are single classification rules (or disjuncts), which are expressed as

$$r_i : (Condition_i) \rightarrow y_i. \tag{4.1}$$

The right-hand side of the rules ("rule consequent") contains the predicted class y_i, and each condition or precondition on the left-hand side ("rule antecedent") contains a conjunction of different attribute tests, such as

$$Condition_i = (A_1 * val_1) \wedge (A_2 * val_2) \wedge \cdots \wedge (A_i * val_i), \tag{4.2}$$

where A_i is an attribute specifier, val_i the value of attribute A_i, and $*$ is one of the set of logical operators $\{=, \neq, <, >, \leq, \geq\}$. A rule r_i covers an object if the attributes of the object satisfy the condition of the rule. A very simple example of such a rule that covers an object with two attributes *weather* (value: hot) and *temperature* (value: 31) may look like

$$r_1 : (weather = sunny) \land (temperature > 28°C) \rightarrow icecream. \tag{4.3}$$

The quality of a classification rule can be evaluated using the measures coverage and accuracy. The *coverage* of a rule is the fraction of records that satisfy the antecedent of a rule, while the *accuracy* of a rule is the fraction of records that satisfy both the antecedent and consequent of a rule (over those that satisfy the antecedent). Both coverage and accuracy of a rule measure the quality of a single classification rule instead of the complete set of rules, and are thus not comparable to performance metrics such as the TP and FP rate, recall or precision (cf. Section 2.4.5). Moreover, the *length* of a rule indicates the number of attributes used within that rule. Some rules may not be mutually exclusive, i. e., more than one rule may cover the same instance, and the rules may predict conflicting classes. If a strict enforcement of mutual exclusiveness is not possible, rules can be ordered in a decision list in decreasing order of their priority. This priority can be defined in several ways, for example, based on the coverage or accuracy of rules, or the order in which rules are generated. A test record is then classified by the highest-ranked rule that covers the record, other rules that also cover this record are not processed. If rules are not ordered, it is also possible to allow a record to trigger multiple rules and consider the consequent of each triggered rule as a (sometimes weighted) vote for that particular class.

Construction process. As already mentioned, it is possible to generate a initial set of rules directly from the data or indirectly from other classification methods such as decision trees. After creating the initial set, the rules are pruned, simplified and sometimes ordered and further optimized. Then, a rule set S_R is created. One of the most well-known rule-based algorithms is RIPPER [Coh95], a learner that forms rules through a process of repeated growing and pruning. During the first phase – the growing phase – rules are made more restrictive in order to match the training objects as accurately as possible. In the succeeding pruning phase, the rules are made less restrictive in order to avoid overfitting of the model.

General evaluation. Rule-based learners are nearly as expressive as decision trees, they are also easy to interpret and to generate, and their performance is often comparable to decision trees (cf., for example, [WF05]). Moreover, the classification speed of rule-based algorithms and decision trees is comparable, but the learning process for rule-based learners is often more time consuming. Contrary to decision trees, rule-based learners cannot deal directly with continuous data and their tolerance to noise and irrelevant attributes is lower (cf. [AC99]).

Relevant literature. As for k-nearest neighbor and decision trees, the principles of rule-based learners are described in almost all introductory data mining and machine learning books. Moreover, there are several review articles [Gom04, SW07], and, as already mentioned, the survey in [Fue99]. Besides the RIPPER method (cf. [Coh95]), there are other well known algorithms such as 1R [Hol93], IREP [FW94], or CN2 [CB91].

4.5 Support Vector Machines

Support vector machines (SVMs) are some of the most recent and most successfully applied classification algorithms and were introduced to the machine learning community by Cortes and Vapnik [CV95, Vap99]. The goal of an SVM is to minimize an upper bound on its expected classification error. SVMs seek the optimal separating hyperplane between two classes by maximizing the margin between the classes, hence, they are also referred to as *maximum margin classifiers.* If an n-dimensional dataset is linearly separable in the n-dimensional space, it is possible to find an infinite number of hyperplanes (a point in the 1-dimensional space, a line in the 2-dimensional space, a plane in the 3-dimensional space) that separate the data points.

Figure 4.2 shows two possible decision boundaries with different margins between the classes. Since decision boundaries with small margins (B_2 in Figure 4.2) are susceptible to model overfitting (a slight change to the decision boundary can have a significant impact on the classification) and tend do generalize poorly on previously unseen examples (cf. [TSK05]), the goal is to find the decision boundary with the maximal margin (B_1 in Figure 4.2). Such a hyperplane, which maximizes the distance to the closest points from each class is called *maximum margin hyperplane.* The closest points to the hyperplane are called the *support vectors.* Only these points influence the position of the hyperplane – all the other vectors (i. e., objects) are not considered.

Although the concept of a maximum margin hyperplane only applies to the problem setting of classification, some support vector machine algorithms have been also developed for regression problems (i. e., numeric prediction, see Section 2.4.4, cf., [SS04]).

Linear SVM. Each example is denoted as a tuple $(\mathbf{x}_i, y_i)$ $(i = 1, 2, \dots, N)$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})^\top$ corresponds to the attribute set for the i^{th} example, and

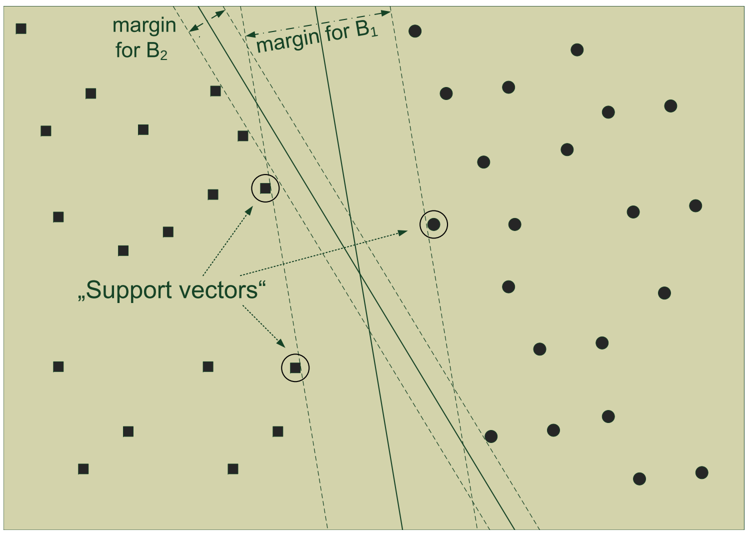

Figure 4.2: Maximum margin between groups.

$y_i \in \{-1, 1\}$ corresponds to its class label. When the dataset is linearly separable, the decision boundary can be written as $\mathbf{w} \cdot \mathbf{x} + b = 0$, where the normal vector $\mathbf{w}$ and the bias (or offset) b are parameters of the model, and $\mathbf{x}$ is the set of attributes of the model. The decision rule is then given by $f_{w,b}(\mathbf{x}) = sgn(\mathbf{w}^\top \mathbf{x} + b)$, i. e., $y_i = sgn(\mathbf{w}^\top \mathbf{x}_i + b)$. During the training phase of SVM, the parameters $\mathbf{w}$ and b of the decision boundary are estimated from the training data, and must fit the following conditions:

$$\begin{aligned} &\mathbf{w} \cdot \mathbf{x}_i + b \geq 1, \text{if } y_i = 1, \\ &\mathbf{w} \cdot \mathbf{x}_i + b \leq 1, \text{if } y_i = -1. \end{aligned} \tag{4.4}$$

The optimization problem for a dataset with N training samples can then be written as:

$$\min_{w} \frac{||\mathbf{w}||^2}{2} \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, ..., N. \tag{4.5}$$

Since this is a convex optimization problem it can be solved using the standard Lagrange multiplier method. First, the optimization problem (4.5) is re-written in the following form

$$L_P = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N} \lambda_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1), \tag{4.6}$$

which is known as the primary Lagrangian for the optimization problem. After minimizing the primary Lagrangian (by setting the derivative of L_P with respect to $\mathbf{w}$ ($\frac{\delta L_P}{\delta \mathbf{w}}$) and b ($\frac{\delta L_P}{\delta b}$) to zero) the optimization problem can then be transformed into a dual optimization problem

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \tag{4.7}$$

which can be solved using numerical techniques such as quadratic programming. A detailed mathematical discussion of this optimization problem is beyond the scope of this thesis and can be found, for example, in [Vap99, CS00, SS01, TSK05, Iva07, Kot07].

The **soft margin method** (cf. [CV95]) is an extension of linear SVMs that allows for mislabeled examples of the training data. If no hyperplane can separate the examples of two classes, this method will choose a hyperplane that splits the objects as good as possible. The soft margin method introduces positive slack variables ξ_i, $i = 1, \ldots, N$ in the constraints of Equation (4.5), which then become

$$\min_{w} \frac{||\mathbf{w}||^2}{2} + C \sum_{i=1}^{N} \xi_i \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., N, \tag{4.8}$$

where C is a constant that appears only as an additional constraint in the primary Lagrange multipliers. The dual Lagrangian for the soft margin method remains unchanged compared to Equation (4.7), except for the restriction that all λ_i are greater than or equal to 0, and are smaller than or equal to C $(0 \leq \lambda_i \leq C)$. Mathematical details are again explained in [Vap99, CS00, SS01, TSK05, Iva07, Kot07].

Nonlinear SVM. The maximum margin hyperplane is able to classify data that are – maybe with some exceptions – linearly separable. Unfortunately, many real-world problems involve data where no useful separating hyperplane can be found. In such cases, SVMs transform the data and map objects from their original coordinate space ("input space") $\mathbf{x}$ into a new larger space ("feature space") $\phi(\mathbf{x})$, where the objects are then linearly separable. This transformation is usually denoted as: $\phi : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}, \mathbf{x} \mapsto \phi(\mathbf{x})$, where $d_1 < d_2$. Figure 4.3 shows a simple case where the data is not linearly separable in its original (2-dimensional) input space. After mapping (ϕ) the data to the 3-dimensional feature space, the objects can be linearly separated.

Using a transformed feature space of sufficiently high dimensionality, any consistent training set can be made linearly separable, and this linear separation in the transformed feature space corresponds to a nonlinear separation in the original input space (cf. [Kot07]). By replacing the original coordinate space in $\mathbf{x}$ in the constraints of Equation (4.5) with its corresponding feature space $\phi(\mathbf{x})$, the optimization

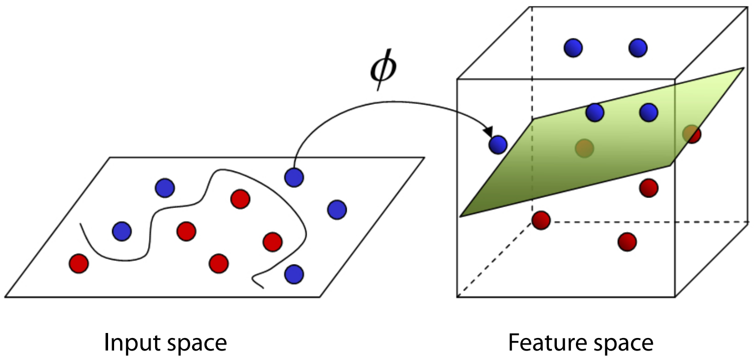

Figure 4.3: Mapping to higher dimensional space (following [TN06]).

problem can now be formulated as:

$$\min_{w} \frac{||\mathbf{w}||^2}{2} \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b)\geq 1, \quad i=1,2,....,N. \tag{4.9}$$

The dual Lagrangian for the constrained optimization problem then becomes

$$L_D=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j), \tag{4.10}$$

Since $\phi(\mathbf{x})$ usually has a much higher dimension than $\mathbf{x}$, maximizing the target function (4.9) and evaluating the new decision function then requires the computation of dot products $\langle\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)\rangle$ in a high-dimensional space. Such computation can be quite expensive and may suffer from the curse of dimensionality (cf. Section 2.4.3).

SVMs overcome this problem by applying a technique known as the *kernel trick*, by choosing a kernel K such that: $K(\mathbf{x}_i\cdot\mathbf{x}_j)=\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$. Kernels are special non-linear functions that have the advantage of operating in the input space, where the solution of the classification problem is a weighted sum of kernel functions evaluated at the support vectors. The kernels allow inner products to be calculated directly in the feature space, without performing the mapping described above [SS01]. Kernel functions are very powerful and allow SVM models to perform separations even with very complex boundaries. The theory behind the kernel trick is based on Mercer's theorem [CV95], which ensures that the kernel functions can always be expressed as the dot product of two input vectors in some high-dimensional space.

Some examples of basic kernel functions are the linear kernel $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i\cdot\mathbf{x}_j$, the polynomial kernel $K(\mathbf{x}_i,\mathbf{x}_j)=(\mathbf{x}_i\cdot\mathbf{x}_j+1)^p$, or the radial basis kernel $K(\mathbf{x}_i,\mathbf{x}_j)=e^{||-\mathbf{x}_i-\mathbf{x}_j||^2/2\sigma^2}$. Since there is no theoretical tool to predict which kernel will give the best results for a given dataset, experimenting with different kernels is often the only way to identify the best function (cf. [Iva07]).

Multi-class SVM. The standard SVM algorithm is only directly applicable for binary, two-class tasks, but SVMs can also be applied to problems with three or more classes.

Besides *multi-class ranking SVMs*, where the SVM decision function attempts to classify all classes (by finding a decision boundary that is able to separate between more than two classes), there are two possibilities to construct multi-class classifiers from combinations of several binary classifiers. In the *one-against-all* classification, one binary SVM for each class is created to separate members of that class from members of all other classes. In the *pairwise classification*, there is one binary SVM for each pair of classes to separate members of one class from members of the other class. Relevant literature for multi-class SVMs can be found, for example, in [CS02, Abe03, PBC06].

General evaluation. Various studies have shown that SVMs are usually able to achieve very good classification results compared to other supervised machine learning algorithms (cf., for example, [BTH01, GDD01, FC04, Kot07, CKY08]). The good classification results are mainly based on the fact that SVMs are not prone to getting stuck in local minima (cf. [SC04]). On the contrary, the possibly biggest limitations of SVMs lie in the questions related to choice of the kernel and in their algorithmic complexity (cf. [Bur98]). While SVMs are able to perform the classification process very fast, their speed of learning is usually much slower than for decision trees, rule-based learners or other learning algorithms. Like decision trees, SVMs can deal well with irrelevant attributes, but the resulting model is usually very hard to comprehend and interpret by domain experts.

Relevant literature. There are several books and survey papers that provide a good introduction and detailed information about SVMs and their mathematical background. Besides the introductory works by Cortes and Vapnik [CV95, Vap99], the survey by Burges [Bur98], two books by Crisianini and Shawe-Taylor [CS00, SC04] as well as the comprehensive book by Smola and Schölkopf [SS01] provide extensive tutorials on support vector machines and the principles of learning with kernels. One of the most cited publications about SVMs (applied to the task of text categorization) is [Joa98]. Huang *et al.* [HKK06] focus on kernel-based algorithms for huge dataset, and Ivanciuc [Iva07] reviews applications of SVMs in Chemistry. References for mathematical background information about SVMs and multi-class SVM are already listed in the appropriate paragraphs. Relevant literature and further references for support vector regression can be found, for example, in the tutorial by Smola and Schöllkopf [SS04].

4.6 Ensemble Methods

Ensemble methods or *classifier combination methods* aggregate the predictions of multiple classifiers into a single learning model. Several classifier models (called "weak" or "base" learners) are trained and their results are usually combined through a voting or averaging process. The idea behind ensemble methods can be compared to situations in real life. When critical decisions have to be taken, often opinions of several experts are taken into account rather than relying on a single judgment. Ensembles have shown to be more accurate in many cases than the individual classifiers, but it is not always meaningful to combine models. Ideal ensembles consist of classifiers with high accuracy which differ as much as possible. If each classifier makes different mistakes, the total error will be reduced, if the classifiers are identical, a combination is useless since the results remain unchanged. Combining multiple models depends on the level of disagreement between classifiers and only helps when these models are significantly different from each other.

4.6.1 Bagging

The principal idea of bagging (short for **b**ootstrap **agg**regat**ing**) is to aggregate predictions of several models of a given weak learner fitted to bootstrap samples of the original dataset by a majority vote (cf. [Bre96]). *Bootstrapping* refers to resampling with replacement from the original sample of instances (not attributes), and a *weak learner* is some pre-defined standard learning algorithm (e. g., a decision tree, SVM,...). The term "weak learner" is widely used across the literature, although weak learners are usually not bad classifiers.

Bagging proceeds as follows: First, M bootstrap samples L_m, m=1,...,M from the original learning sample L are drawn. Second, a base learner is applied for each of the bootstrap samples $b_m(L_m)$, and the classifier models $f(b_m(L_m))$ are constructed. In the classification process, the individual models are aggregated using weights $a_m = 1/M$, yielding the ensemble (cf. [HDB05])

$$\sum_{m=1}^{M} b_M(L) = \frac{\sum\limits_{m=1}^{M} f(b_m(L_m))}{M}. \tag{4.11}$$

As can be seen, in the standard bagging procedure each sample model (i. e., weak learner) receives equal weight. The standard algorithm for bagging applied to the task of classification is shown in Algorithm 6 (cf. [WF05]).

Some main advantages of bagging are its ability to reduce variance and to avoid model overfitting (cf. [Kot07]). It is an intuitive and easy to implement approach

Algorithm 6 – Pseudo-code for ensemble classification using bagging.

```
Model generation
    Let $N$ be the number of instances in the training data, and
        $M$ be the number of samples to be drawn.
    For each of the $M$ samples:
        Sample $N$ instances with replacement from training data.
        Fit the weak learner to the sample and construct classifier.
        Store the resulting model.
.....................................................................
Classification
    For each of the $M$ models (samples):
        Predict class of new, unclassified instance using model.
    Return class that has been predicted most often.
```

and can be used with any learning algorithm as weak learner. Moreover, bagging has shown to achieve a very good performance in several studies and the empirical fact that bagging improves the predictive performance of several learning algorithms is widely documented (cf. [Bre96, ZTD01, PY02, ES05]). However, it is computationally rather expensive (no explicit feature selection) and often lacks the possibility of interpretation of the resulting model. Since bagging exploits the instability inherent in learning algorithms (i. e., here, the term *instability* refers to the fact that different samples lead to different models), it is only useful if weak learners are unstable.

4.6.2 Random Forest

A random forest [Bre01] is a special modification of bagging that mixes the bagging approach with a random sub-sampling method. While bagging works with any algorithm as weak learner, random forests are ensembles of *un*pruned classification or regression trees (Section 4.3). The commonly used growing algorithm for the single decision trees used within the random forest algorithm is CART (cf. Section 4.3). Just like bagging, random forests select instances randomly with replacement (bootstrap), but unlike bagging, random forests also sample attributes (without replacement) for each tree. The trees are grown to maximal depth (no pruning) and each tree performs an independent classification/regression. Then each tree assigns a vector of attributes or features to a class and the forest chooses the class having most votes over all trees (using a majority vote or averaging).

Each tree is grown as follows: If the number of cases in the training set is N, sample N cases at random with replacement (i. e., the size of the sample is equal to the size of the training set – but some instances of the training set may be missing in the sample while some other instances may appear multiply in the sample). This sample is the training set for growing the tree. If there are M input variables, a number $m << M$ is specified such that at each node m variables are selected at random out of the M and the best split on these m attributes is used to split the node. The value of m is held constant during the forest growing. Each tree is grown to the largest extent possible, there is no pruning (cf. Section 4.3). The standard algorithm is shown in the pseudo code in Algorithm 7.

Algorithm 7 – Pseudo-code for random forest.

```
Model generation
    Let $N$ be the number of instances in the training data, and
        $M$ be the number of samples to be drawn (i. e., number of trees).
    For each of the $M$ samples:
        Sample $N$ instances with replacement from training data.
        Sample $m << M$ attributes without replacement.
        Fit a DT to the sample and construct classifier/regression model.
        Store the resulting model.
.......................................................................
Classification / Regression
    For each of the $M$ decision trees models (samples):
        Predict class/probability of new, unclassified instance using model.
    Return class that has been predicted most often (classification) or use
    an averaging process (regression).
```

Similar to bagging, random forests are easily comprehensible and can reduce the variance of the prediction accuracy, but due to the sampling of attributes, the learning process of random forests is usually faster. Random forests can handle a very large number of input variables, and even when a large portion of attribute values is missing, it is often able to maintain the desired accuracy. Moreover, it is possible to measure attribute importance as well as the similarity between attributes. The predictions of random forests have the drawback that they are the outcome of a black box, especially if a small number of informative variables are hidden among a great number of noisy variables. Random forests are prone to overfitting if the data is noisy (cf. [Seg04]), and the CART algorithm for growing the single trees within random forests does not handle large numbers of irrelevant attributes as well as decision tree algorithms that use entropy-reducing splitting criteria, such as C4.5 (cf. [GGM08]).

4.6.3 Boosting

In the ideal bagging situation, all models complement each another, each being a specialist in a part of the domain where the other models do not perform very well. Boosting [FS97, Sch03] exploits this idea by explicitly seeking models that complement each other, and applies certain re-sampling strategies in order to get the most informative strategic data. Instances that were classified incorrectly by some classifiers are taken into the training set more often, i. e., boosting encourages new models to become experts for instances handled incorrectly by earlier models. The general boosting algorithm (following [WF05]) is shown in Algorithm 8.

Algorithm 8 – Pseudo-code for boosting.

```
Model generation
    Assign equal weight to each training instance.
    For each of $t$ iterations:
        Select weighted dataset based on randomness and instance weights.
        Apply learning algorithm to weighted dataset and store model.
        Compute training error $e$ of the model on weighted dataset.
        If ($e == 0$) or ($e >= 0.5$):
            Terminate model generation.
        For each instance in dataset:
            If instance is classified correctly by model:
                Multiply weight of instance by $e/(1-e)$ (weight is decreased)
        Normalize weight of all instances.
..........................................................................
Classification / Regression
    Assign weight zero to all classes.
       For each of the $t$ models:
    Add $-log(e/(1-e))$ to weight of class predicted by the model.
    Return class with highest weight
```

On the one hand, boosting is similar to bagging because both methods combine models of the same type (i. e., same weak learner) and use voting (for classification) or averaging (for regression) to combine outputs of individual models. On the other hand, boosting differs from bagging because it is an iterative process where each new model is influenced by the performance of the models built previously. Moreover, boosting weights a model's contribution by its performance rather than giving equal weight to all models (as done with bagging). The main advantage of boosting is the fact that combined classifiers can be built even from very simple classifiers as long as they achieve less than 50% error. Boosting often produces classifiers which are significantly more accurate on fresh data than ones generated by bagging. Nevertheless, there are several studies (for example, [Die02, MR03, BH08]) that have shown that boosting sometimes fails in practical situations where it generates a classifier that is significantly less accurate than a single classifier built from the same data. This behavior usually indicates that the combined classifier overfits the data, which is one of the major drawbacks of boosting. Moreover, standard boosting is sensitive to noise (cf. [BS03]). Adaptive Boosting (AdaBoost, cf. [Sch03]) is a famous boosting variant that aims at reducing the susceptibility of boosting to the overfitting problem. Although AdaBoost was reported to be sensitive to noisy data and outliers, it has achieved popularity because of its good generalization and the fact that it may handle even very large sets of features (compared to other boosting variants). Some comments about the convergence of AdaBoost can be found in [RSD07].

4.6.4 Stacking

Although Stacking (stacked generalization, [Wol92]) is older than bagging and boosting it is less widely used. Unlike bagging and boosting, stacking combines models built by *different* learning algorithms. Instead of voting or averaging the results of different models or choosing the best model, stacking uses the concept of a *meta-learner*. Stacking tries to learn which weak learners are the reliable ones, using another learning algorithm (the meta-learner) to discover the best way to combine the outputs of the base learners. If applied correctly, the meta-learner can be more efficient than a standard voting or averaging procedure. Since stacking does not rely only on one specific weak learner but on a combination of various weak learners, it is less prone to overfitting, produces a more general model of the data, and has been shown to be consistently effective for applications in several data mining domains [SPS05]. Unfortunately, stacking is difficult to analyze theoretically and there is no generally accepted best way of doing it. Moreover, it is hardly comprehensible and is often considered as the ultimate black box amongst the group of ensemble learners.

4.6.5 General Evaluation

Ensemble methods are often able to improve accuracy by combining the predictions of a set of different hypotheses. However, Ferri [FHR02] annotates two important shortcomings associated with ensemble methods: (*i*) huge amounts of memory are required to store a set of multiple hypotheses and, more importantly, (*ii*) the comprehensibility of a single hypothesis is lost. Other studies report that these shortcomings are true for both, supervised and unsupervised learning methods such as clustering (cf. [VJ06]). Due to the nature of ensemble methods, most algorithms are not prone to overfitting (except boosting in some cases), and can deal with noisy or missing attribute values. Among the four ensemble methods mentioned in this section, bagging and random forest are often preferred over boosting and stacking, but there is no general proof of which method is best (cf. [VD02]).

Relevant Literature for Ensemble Methods. Besides the introductory work for bagging, random forests, boosting and stacking, there are some surveys of ensemble methods [Die00, Die02], as well as some work with general comparisons of ensemble methods (cf. [VD02, OT08]). Bagging has been combined successfully with cost-sensitive learning ("MetaCost", [Dom99]) and boosting [Rid02, Det04]. [PY02] introduced two variants of bagging: *subsample aggregating*, where sub-sampling is used instead of the bootstrap for the aggregation, and *bragging* (bootstrap robust aggregating), which uses the sample median over the bootstrap estimates instead of the sample mean.

Various studies on random forests have shown that the performance of decision trees could be improved if ensembles of trees were used (cf. [Bre01, Die02, SLT04]). Svetnik *et al.* [SLT03] have combined random forests with a feature selection algorithm based on measuring the importance of single features, and successfully applied this combination to the task of QSAR-modeling (cf. Section 5.2). [PP08] used random forests for multi-class classification and regression by combining it with multi-nominal logit (MNL), a generalization of logistic regression that allows more than two discrete outcomes and is commonly applied within the customer relationship management domain. Recently, [BDL08] discussed several consistency theorems for various versions of random forests and other randomized ensemble classifiers, and [AN09] used random forests to uncover bivariate interactions in high dimensional small datasets.

Interesting recent work on boosting comprises a study based on boosting with noisy data [Jia04], and the publication of a new boosting algorithm for improved time-series forecasting based on neural networks [ABC08]. An empirical evaluation of several state-of-the-art methods for constructing ensembles of heterogeneous classifiers with stacking and a comparison to single classifiers can be found in [DZ04]. The results show that stacking performs best by selecting the best classifier from the ensemble by cross validation. Another article [RPN06] investigates an algorithmic extension to stacking that prunes the ensemble set and achieves equal classification accuracy while having the advantage of producing smaller and less complex ensembles.

4.7 Vector Space Model

In information retrieval, the vector space model (VSM, cf. [SWY75, DDF90, RW99]) is a widely used model for representing information. Each item or document is represented as a potentially very high dimensional vector whose components reflect a term or key word associated with the given document. Each document can thus be represented as a vector in $\mathbb{R}^t$, where each axis represents a term. The value of the components is typically a function of the frequency with which the term occurs in the complete document collection or in a single document (cf. [Dum91]). A matrix containing d documents described by t terms is represented as a $t \times d$ term-by-document matrix. The distance between the document vectors and a query vector is the basis for the information retrieval process, which is summarized very briefly in its standard form in the following (for a more detailed description see, for example, [Lan05]).

Generally speaking, a vector space model for d documents is based on a certain set T of t terms. Then, the value v_{ij} for each of these terms t_i from each document j has to be extracted. The t-dimensional (column) vector $\mathbf{v}_j$ with the components $v_{ij}, i = 1, 2, \ldots, t$, then represents document j in the VSM. The $t \times d$ matrix A is composed using all the vectors $\mathbf{v}_j$ as columns, and represents the training samples against which the query vectors are compared.

Query matching. Given a query vector $\mathbf{q}$ of length t, the distances of $\mathbf{q}$ to all documents represented in A can then be measured (for example) in terms of the cosines of the angles between $\mathbf{q}$ and the columns of A, but other similarity measures are also possible (see, for example, the review in [JF87]). The column with the smallest angle (largest cosine value) with the query vector represents the closest match between the document collection and the query vector. The cosine φ_i of the angle between $\mathbf{q}$ and the i-th column of A can be computed as

$$(\text{VSM}) : cos\varphi_i = \frac{e_i^\top A^\top q}{||Ae_i||_2||q||_2}, \tag{4.12}$$

where $||x||_2$ is the Euclidean vector norm defined as $||x||_2 = \sqrt{x^\top x}$.

Many text collections contain documents having different contexts. In this case, the number of terms is often much greater than the number of documents, such that $t >> d$. Since documents generally use only a small subset of the entire dictionary of terms generated for a complete term-by-document matrix, most elements of A are zero (cf. [BDJ99, Lan05]).

When comparing the vector space model with state-of-the-art supervised learning algorithms (as the ones presented earlier in this chapter), the strong relationship to the kNN model is obvious. VSM can also be considered as an instance-based learner – in the learning phase, the data is only encapsulated from the training data, the process of learning is postponed until the classification process. The main difference between kNN and VSM is the applied distance measure.

General evaluation. A problem of the VSM is that it may fail if terms have multiple meanings (e. g., issue, light) or if different words (i. e., tokens) exist that have the same meaning (e. g., car, automobile, . . .). *Polysemy* (words having multiple meanings) and *synonymy* (multiple words having the same meaning) are two fundamental problems when retrieving relevant information from a database. The closeness of a query vector to documents concerning the same topic might be returned as not relevant if polysemic or synonymic words are used. Various approaches have been developed to respond to such failures (e. g., usage of a controlled vocabulary). The most important method aims at replacing the exact term-by-document matrix by an approximation of lower rank in order to uncover latent information and connections between different terms. This approach is known as *latent semantic indexing* and is explained in Section 4.8.

Relevant literature. The idea of using a vector space model for representing and classifying documents was presented in an article by Salton *et al.* [SWY75], which is partly based on earlier work by Sparck-Jones [Jon72]. A detailed introduction to the vector space model can be found in a variety of scientific papers or books, for example, [SM86, BDJ99, Dum91, SS95]. Raghavan and Wong [RW99] give a critical analysis of the vector space model for information retrieval. Besides its classical application area, document classification, vector space models have recently been used, for example, for ontology-based IR [CFV07], XML-based IR [GCL09] or protein retrieval [AAR07].

4.8 Latent Semantic Indexing

Latent semantic indexing (LSI, c.f., for example, [DDF90, BDJ99, Lan05]) – which is also referred to as *latent semantic analysis* (LSA) – is a variant of the basic vector space model. Instead of the original matrix A, the singular value decomposition (c.f. Section 3.3.3) is used to construct a low-rank approximation A_k of A such that $A = U\Sigma V^\top \approx U_k\Sigma_k V_k^\top =: A_k$. LSI works by omitting all but the k largest singular values of the SVD decomposition. When A is replaced by A_k, then the cosine φ_i of the angle between $\mathbf{q}$ and the i-th column of A is approximated as

$$cos\varphi_i \approx \frac{e_i^\top A_k^\top q}{||A_k e_i||_2||q||_2}, \tag{4.13}$$

When A_k is replaced with its decomposition $U_k \Sigma_k V_k^\top$, (4.13) becomes

$$cos\varphi_i \approx \frac{e_i^\top V_k \Sigma_k U_k^\top q}{||U_k \Sigma_k V_k^\top k e_i||_2 ||q||_2}, \tag{4.14}$$

At this step it is possible to identify parts of the Equation (4.14) that only need to be computed once and can be reused for each query matching step. The left part of the enumerator can be pre-computed, and the result of $e_i^\top V_k \Sigma_k$ can be stored in h_i. In the denominator, the left norm can be computed prior to the query matching. Moreover, since U_k and V_k are both orthogonal matrices, $||U_k \Sigma_k V_k^\top e_i||_2$ can be replaced with $||\Sigma_k e_i||_2$ (since $||X||_2$ and $||X^\top||_2$ are both equal to 1 if X is orthogonal). Equation (4.14) thus becomes

$$cos\varphi_i \approx \frac{h_i U_k^\top q}{||\Sigma_k e_i||_2 ||q||_2}. \tag{4.15}$$

At runtime, only $||U_k^\top q||$ and $||q||_2$ must be computed, thus saving computational cost for each query.

Analysis of LSI. The LSI subspace captures the essential meaningful semantic associations while reducing redundant and noisy semantic information. LSI goes beyond lexical matching and aims at addressing the problem of using individual keywords to identify the content of documents. LSI is based on the principle that words that are used in the same contexts tend to have similar meaning. In other words, queries against a set of documents that have undergone LSI will return results that are conceptually similar in meaning to the search criteria even if the results do not share specific words with the search criteria (cf. [DDF90]). The approximated data A_k often gives a cleaner and more efficient representation of the relationship between data elements (cf. [LMA06]) and helps LSI to overcome the problems of polysemy and synonymy. The goal is to capture the underlying (latent) semantic structures, which often index the documents better than individual indexing terms.

General evaluation. From a computational point of view, LSI also has advantages over the basic VSM in terms of storage requirements and computational cost for determining the distances between the query vector **q** and the documents (see Equation (4.14)). The storage saving, discussed in Section 3.3.3, can be achieved by a representation based on the three small decomposition matrices within the SVD instead of the original term-by-documents matrix. These considerations are discussed in greater detail in [BDJ99]. One of the main drawbacks of LSI is the rather large computational cost of computing the singular value decomposition. Although there are relatively fast algorithms for computing only the first k singular values of an SVD (cf. Section 3.3.3), k must be chosen properly in order to avoid recomputing the SVD (if k turns out to be too small). Actually, determining the optimal number of rank of the low-rank approximation (value of k for SVD) is a main challenge of LSI. A higher rank enables more specific comparisons of concepts contained in a collection of text, while a lower rank allows for broader comparisons of these concepts (better to cope with polysemy and synonymy). A recent research study [Bra08] has demonstrated that a rank around 300 will usually provide the best results for a document collection of moderate size, for larger collections (millions of documents) the rank should be increased to 400. However, these numbers should be taken only as indications since the best choice of k is usually problem-dependent.

Relevant literature. Besides its classical application area text mining [DDF90, Lan05, BB05b], LSI has been used successfully in other areas such as cross lingual applications [LL90, LDL97], spam filtering [Gee03, GJL07], gene clustering [HHW05], information visualization [LLD04], calculating chemical similarity for drug discovery problems [HFS01, HSN01], and several different other application areas. The effectiveness of LSI has been demonstrated empirically in several studies. A work by [PTR98] goes beyond empirical evidence and provides a theoretical evidence that LSI is able to deal effectively with the problem of synonymy.

4.9 Other Relevant Classification Methods

In this section, several classification methods that are referred to in Chapter 5 are summarized briefly. A detailed survey of all available learning algorithms is beyond the scope of this thesis, but can be found in the relevant literature given in Chapter 2, for example, in [Dun02, TSK05, HK06, CPS07, Lar07, ML07].

4.9.1 Naïve Bayesian

A naïve Bayesian (NB) classifier (cf., for example, [TSK05, WF05]) is a simple statistical modeling approach based on applying Bayes' theorem from Bayesian statistics. It estimates the class-conditional probability by assuming that the attributes are equally important and conditionally independent, by assuming that the presence or absence of a particular feature is independent to the presence or absence of any other feature. NB is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, naïve Bayesian classifiers are often able to achieve very high classification results (cf. [KP04]).

4.9.2 Neural Networks

The study of (artificial) neural networks (ANN/NN, cf. [KS93]) was inspired by attempts to simulate biological neural systems, more precisely the behavior of the human brain. The human brain can learn by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse (cf. [Roj96]). Analogous to this structure, ANNs are composed of an interconnected assembly of nodes and directed links. The training of ANNs is usually very time consuming and ANNs are also sensitive to the presence of noise in the training data (cf. [TSK05]). Contrary to that, their classification speed is rather fast. Moreover, ANNs can handle redundant features and are able to deal with discrete/binary/continuous attributes (cf. [Kot07]).

The **perceptron** is the simplest representative of an ANN. It consists of two types of nodes – several input nodes and one output node. Each input node is connected via a weighted link vector to the output node, these weighted links are used to emulate the strength of a synaptic connection between neurons.

A **multilayer ANN** (or *multilayer perceptron*) has a much more complex structure than a perceptron model, which is caused by additional intermediary (*hidden*) layers between input and output layers, and by the variability of activation functions that can be used. The nodes embedded in these layers are called hidden nodes. The nodes in one layer can either be connected only to nodes in the next layer, or may be connected to nodes within the same layer and nodes from the next layer. The first approach is called a feed-forward network, the latter one is called recurrent neural network.

Self-organizing (SOM) or Kohonen maps (cf. [Koh00]) are special types of neural networks and are counted among the group of unsupervised classification methods, i. e., they work without the class information of a sample (there is no a priori known output at the time of classification). SOMs generate a topology-preserving non-linear mapping of a high-dimensional input space (usually the descriptor space) to a low-dimensional space (cf. [Sch00]).

Contrary to SOMs, **counterpropagation neural networks** (CPG-NN, cf. [Hec87]), a similar, but slightly different class of neural networks, represent a two-step learning algorithm, which can be divided into a unsupervised (Kohonen) learning step, and a consecutive supervised, prediction step. Since each input pattern of a CPG-NN needs a unique node in the hidden layer, this type of network is very limited to be applied for real world problems (cf. [FC08]).

4.9.3 Discriminant Analysis

Discriminant analysis (cf., for example, [McL04]) is usually used to either assess the adequacy of classification given the group memberships of the objects under investigation, or to assign an object to one out of several a priori known groups of objects. **Stepwise discriminant analysis** (SDA, cf. [Jen77]) is often used to identify redundant variables and to select variable subsets that preserve multivariate data structure adequate for discrimination purposes. SDA tries to iteratively modify a candidate feature subset (instead of analyzing all dimension subsets exhaustively) until no further improvement is possible (starting with an initially empty set of features).

Chapter 5

Application Areas

All experimental evaluations in this thesis are performed on data from either one or both application areas mentioned in this chapter. On the one hand, we consider email filtering, where the feature space contains various properties of email messages derived from the header of the message, from the body of the message, or from both. Besides the well known problem of spam filtering, we also discuss the problem of filtering potentially harmful phishing email. On the other hand, we consider drug discovery problems where quantitative representations of molecular structures are encoded in terms on information-preserving descriptor values. More precisely, we focus on the application of *QSAR modeling* (quantitative structure activity relationship).

In the following, the classification problems arising in the contexts of email classification and QSAR modeling are discussed briefly. Moreover, relevant literature is reviewed for both application areas and the performance metrics from Section 2.4.5 are adapted to the properties of the specific classification tasks for each application field.

5.1 Email Filtering

About a decade ago, unsolicited bulk or commercial email (UBE, UCE, "spam") started to become one of the biggest problems on the Internet. In [Cor07], email spam is defined as *"unsolicited, unwanted email that was sent indiscriminately, directly or indirectly, by a sender having no current relationship with the recipient"*. The transmission of spam email has several negative consequences (cf. [Cor07]). Beside *direct consequences* (loss of money, Trojan horses, cyber attacks) mostly borne by the victims (i. e., individuals fooled by the spam email), *network resource consumption* is an enormous problem caused by spam. The vast majority of email traffic today is spam – estimations from web security service providers indicate a spam rate of up to 90% (cf. [Mes09]). This traffic consumes enormous bandwitdth and storage, and increases the risk of untimely delivered (or even outright loss of) messages. *Human resource consumption* caused by spam is another severe problem, especially for companies. Amongst all consequences caused by spam, the loss of legitimate email messages is often the greatest problem since relevant pieces of information may be lost. Incorrectly classified messages, overlooking of legitimate email in a pool of spam messages, or congestion problems on email servers due to spam are some examples of how email can be lost.

In recent years, *phishing* ("password fishing") has become a severe problem in addition to spam email. The term covers various criminal activities which try to fraudulently acquire sensitive data or financial account credentials from Internet users, such as account user names, passwords or credit card details. The steal of the victims identity is thus a dangerous consequence caused by phishing. Phishing attacks use both social engineering and technical means. Although the phishing rate in the total email traffic is much smaller than the spam rate (0.2-0.4%) (cf. [Mes09]), the damage caused by successful phishig attacks usually exceeds the damage caused by single spam messages considerably. In contrast to unsolicited but harmless spam email, phishing is an enormous threat for all big Internet-based commercial operations. The structure of phishing messages tends to differ significantly from the structure of spam messages, but is usually closer to the structure of regular ham messages (because for a phishing message it is particularly important to look like a regular message from a trustworthy source). As a consequence, phishing email is usually much harder to distinguish from legitimate email than spam email (even for humans).

5.1.1 Classification Problem for Email Filtering

In this section, general performance metrics discussed in Section 2.4.5 are adopted to the problem of email classification. Classical spam filtering is a *binary* classification problem in the sense that every incoming email message has to be classified as "spam" or "not spam". In the context of spam filtering, a "positive" usually denotes a spam message and a "negative" usually denotes a ham message. Consequently, a true positive (TP) is a spam message which was correctly classified as spam, and a false positive (FP) is a ham message which was wrongly classified as spam. The goal is to maximize the rate of true positives and to minimize the rate of false positives simultaneously (in order not to loose possibly important ham messages). Thus, a lower TP rate (with a lower FP rate) is often preferable over a higher TP rate (with a simultaneously increasing FP rate).

As an alternative to classical binary spam filtering, the problem of classifying email into ham, spam and phishing is a *ternary* classification problem. A ternary classification problem can also be addressed as a sequence of two binary classification problems (for example, one filter for separating ham from spam and phishing email, and a second filter for separating spam from phishing email), but this usually involves some drawbacks such as additional overhead, lower efficiency, and, most important, often lower classification accuracy (cf. [GP09]). For a ternary classification, the definitions of TP, FP, TN and FN cannot be adopted, since a ternary classification has nine different possible outcomes (cf. Table 2.2 in Section 2.4.5). Instead, a true positive rate (fraction of correct classification) and a false positive rate (fraction of the sum of all incorrect classifications) for each class can be defined.

5.1.2 Relevant Literature for Spam Filtering

Generally, methods for filtering spam email can be categorized according to their point of action in the email transfer process. Consequently, three classes of approaches can be distinguished. *Pre-send* methods act at the *sender side* before the email is transported over the network, whereas *post-send* methods act at the *receiver side* after the email has been transferred to the receiver. A third class of approaches comprises *new protocols*, which are based on modifying the transfer process itself. Pre-send methods and the development of new protocols are very important because of their potential to avoid most of the waste of resources (network traffic, server workload, etc.) caused by spam. However, the efficiency of pre-send methods and of new protocols heavily depends on their widespread deployment. It is unrealistic to expect global acceptance and widespread use of these new methods in the near future, and thus the third group of methods, post-send spam filtering methods, continues to be the "work horse" in spam defense.

Pre-send methods. The focus of pre-send methods is on potential solutions for detecting and preventing spam in outgoing email traffic on the sender side, i. e., on technology applicable on outgoing mail servers. The two basic strategies that can be distinguished in this category are strategies for increasing the cost of sending email and strategies for increasing the risks of sending spam due to stricter legal regulations and stricter enforcement of these regulations. There are several interesting approaches to increase the senders cost and to make the spammers business model unprofitable, either by introducing monetary fees or by delaying the sending of each email.

The introduction of *monetary fees* for each email relies on the idea to "pay" some amount of (possibly symbolic) currency for each email to be sent (micro payment).

This idea is based on the fact that spammers need to send millions of messages to make profit (since the response rate is very low, for details, see [ISG06]). Even a very small payload would affect the spammers' business model, while the costs for private users and companies remain negligible. The *lightweight currency protocol* (LCP, [TH04]) is an example of such a strategy that proposes a mechanism where servers require payment for accepting incoming messages. Two limiting problems of money based solutions are the relatively high administration overhead and the fact that the very popular free email accounts do not fit into this strategy.

Technical solutions for delaying the sending of each email message are mostly based on CPU time, where the sender is required to compute a moderately expensive function (*pricing function*) before the email is actually sent ("CPU-bound strategies"). The computation of such a function usually takes a few seconds and thus strongly limits the number of spam messages that can be sent from a computer in a specific time period. Regular email users should hardly be influenced by the computation of the pricing function. A software plug-in for mail clients called Hashcash [Bac02] is a well-known representative of CPU-bound strategies. Since the delay caused by the computation of the function is usually not independent of the hardware of the computer system, the lack of fairness of these pricing functions is a major drawback. Some studies have focused on this problem. [DGN03], for example, introduced memory-bound functions which do not rely on hardware performance as much as CPU-bound functions. However, so far no final solution for the fairness problem has been found.

In [GHI05], the authors use a different approach and adapt a token-bucket strategy (which is traditionally used for traffic shaping) to fit the needs of email traffic regulation. The approach is based on creating its own "currency" (tokens) for email messages and includes a starting seed as well as a certain refill rate. Each message sent consumes a certain number of tokens. This implementation allows to impede heavy-duty users while remaining unnoticed by common users.

New protocols. Methods of this group of email filtering strategies affect both, sender and receiver side. Most of these techniques comprise suggestions for renewing or altering email transfer protocols. The *Internet mail 2000* project (IM2000, [Ber00, Boy04]) designed an Internet email infrastructure which is based on the concept that email storage is the sender's responsibility. IM2000 uses a pull mechanism and stores email at the sender's ISP (Internet Service Provider), which becomes the always-on-line post office from which the receiver picks up the message. Although the idea seems interesting, IM2000 would require a global adoption and acceptance of a this new email infrastructure, which is not really expectable. The *authenticated mail transfer protocol* (AMTP, [Wei03]) is being designed as a replacement for the

SMTP protocol (*simple mail transfer protocol*), with security features designed to reduce the impact of spam for receiving email servers. AMTP accepts email only from authenticated senders (using an SSL-like authentication protocol), allows for publishing "policies" and indicates what type of email is being sent. Just as for IM2000, a widespread acceptance for AMTP is – unfortunately – very unrealistic in the near future.

Contrary to IM2000 and AMTP, Greylisting (cf. [Har03]) is based on utilizing properties of the SMTP protocol instead of replacing it. The basic idea is to (temporarily) reject an email message when it is first received and to accept it only when it is resent. This strategy is based on the observation that many spammers use incomplete email server implementations which do not resend a message if it is temporarily rejected. The efficiency of greylisting was evaluated empirically and demonstrated several times (cf. [TWM04, Lev05]). Nevertheless, in its conventional form it has important drawbacks: It introduces delays in the email delivery process, legitimate email may potentially get lost if an email server not conforming with the SMTP standard is used, and difficulties may arise in the process of distinguishing between first and second delivery attempts of email messages, especially when email server farms are involved. Moreover, conventional greylisting is only a short term strategy, because it is quite easy for spammers to adapt to it and to bypass it by sending every email twice.

[JGK08] have proposed the concept of a 2-level greylisting technique to be used in the context of a component architecture (cf. [GJL07]) for detecting and filtering spam. In contrast to previously existing greylisting approaches, this method successfully handles messages originating from server farms and it cannot be bypassed by sending identical messages repeatedly. It has been illustrated that this technique allows for achieving very high spam blocking rates of 99.9% and higher. Moreover, a reputation-based trust mechanism has been integrated into the 2-level greylisting technique which significantly reduces the transfer delay for legitimate messages caused by the additional greylisting level. However, the email messages from untrusted senders still experience a delay between some minutes and a few hours.

defNullSpam [Gar07] is a spam filtering system which uses a strategy similar to that of greylisting by sending an automated response to the point of origin of the incoming unknown email. This automated response contains a code of verification which has to be manually sent back by the recipient of the automated response. Returning this code/key will create a link to this address of origin and enter it in an "approved sender list". This approach requires human interaction to ensure the delivery of email and thus it is difficult to handle legitimate mass email.

Post-send methods. Spam filters of this group act at the receiver side. Most post send methods operate after the email has already been transfered completely to the receiving SMTP server (i. e., after the receiving SMTP server has assumed responsibility for delivering the message), but some of them act already at the beginning of the SMTP dialogue. Black- and whitelisting – two techniques based on the source of email – are able to block/pass a message before it is completely transfered to the receiving mail server. At the beginning of the SMTP dialogue, the receiving SMTP server verifies if the sender's IP address appears on a blacklist or a whitelist, respectively. Any message coming from a source appearing on a whitelist will be accepted and will bypass filtering, while any message coming from a source appearing on a blacklist will (ideally) not be received completely by the receiving SMTP server. Both black- and whitelists can be self-maintained or maintained globally by third-party organizers, such as real time blacklists (RBLs) which can be scanned on-line. Black- and whitelists must be updated regularly and suffer from the fact that either all email from a given host is accepted, or all email is rejected. Two other approaches based on the source of email – policy frameworks [Ioa03] and digital signatures [TH03] – can be used to authenticate the sender of email. Unfortunately, both approaches need a central authentication authority which is difficult to realize in practice.

Most other post-send methods act after the email has been transfered to the receiving SMTP server, and thus tend to be purely *reactive*. Content-based filtering techniques depend on the content of email, and thus can only be applied after the email has been transfered to the receiving SMTP server. Approaches such as fingerprints (e. g. [Sym09]), (digital) signatures (e. g. [TH03]), checksums (e. g. [Sch09]) or URL analysis (e. g. [KLL03]) are examples of content-based email filtering. Another technique, Bayesian spam filtering (cf. [SDH98]), is a statistical filtering technique that is based on the naïve Bayesian classifier (NB, cf. Section 4.9) to separate spam email from legitimate email. The idea is to compute a conditional probability for an email being spam based on the words or tokens it contains. This technique is exclusively based on textual features, making this technique prone to be fooled by diluting the spam message with enough obviously innocent words.

Rule-based filters combine several methods mentioned before and block email based on a pre-determined set of rules. Based on these rules, features describing an email message can be extracted and used as input (training and test data) for several machine learning algorithms. A single email can be considered as an instance, sample or object, and each rule can be considered as a variable, attribute or feature describing an email. Based on some already labeled training data, a classification model is built. Then, a classification process can be applied to predict the class (ham, spam, phishing) of unclassified email. Besides these rules mentioned before, it is also possible to use other attributes for classification, such as purely text-based features, where single tokens are extracted from the content of email messages (cf. Section 6).

A de-facto standard of a rule-based spam filtering system is the SpamAssassin system [McG07, Spa09], which extracts a large number of features from incoming email messages, comprising pieces of information from header, body, and the full text of the message. SpamAssassin uses several mechanisms including text analysis, DNS block-lists, collaborative filtering databases, or a built-in Bayesian classifier, and requires training samples of labeled spam and non-spam email in order to fine tune parameters and optimize learning.

Several machine learning algorithms (some of them mentioned in Chapter 4) have been applied to the task of spam detection. An example for the application of Bayesian spam filtering based on textual features to the scenario of spam filtering is given in [AKC00]. Delany *et al.* [DCC05, DCD05] have studied and assessed a case-based reasoning email classification as a lazy learner method that outperforms naïve Bayesian (NB) classifiers (cf. Section 4.9). Contrary to that, Lai [Lai07] performed a comparative study of the performance of various machine learning methods in spam filtering, and showed that Bayesian classifiers are able to achieve better results than kNN and comparable results to an SVM model. Chuan *et al.* [CXM05] presented an approach based on an LVQ-based (learning vector quantization) neural network that outperforms NB models, and Bratko *et al.* [BCF06] have investigated an approach based on adaptive statistical data compression models and report very good classification results of their empirical evaluation.

Fdez-Riverola *et al.* [FID07] presented an instance-based reasoning email filtering model that partly performed better than classical machine learning techniques (SVM, AdaBoost, NB) and other successful lazy learner approaches in the domain of spam filtering. Youn *et al.* [YM07] proposed a spam filtering method using adaptive ontologies that allows for machine-understandable semantics of the data. A comprehensive survey of several machine learning-based post-send methods can be found in [Cor07], and [Hid05] provides a list of research papers dealing with machine learning methods for building spam filters.

5.1.3 Relevant Literature for Phishing

Compared to spam filtering, only relatively little research has been done so far on specifically detecting phishing email. Chandrasekaran *et al.* [CCU06] have proposed a method that sends faked formular data and analyzes the returned answers. Their work is based on the assumption that phishing sites usually do not check input data, whereas an authentic web site would usually produce an error message for incorrect input data. Phishing toolbars (for example, [McA09, Goo09]) aim to visually warn the user about the web site referred to, but usually act *after* the user has already decided to follow a link provided in an email.

Other approaches are based on the general concept of feature-based phishing detection, where the focus is on analyzing the content of an email message. Obviously, feature-based phishing detection is closely related to rule-based email filtering and thus fits the classification methodology applied in this thesis. In the approach by Liu *et al.* [LDH06], several key words are extracted from every email message. Web pages linked within these messages are then compared with web sites which are close to these key words based on their visual layout. The SpamAssassin system [Spa09] mentioned in Section 5.1.2 can also be used to identify phishing email – recently, some rules which specifically target phishing messages have been integrated into the system. Investigations of special features that are indicators for phishing email were studied in [FST07]. Here, ten features were used for classifying phishing attempts, and are reported to achieve a very high classification accuracy. In another study by Abu-Nimeh *et al.* [ANW07], the phishing detection performance of several machine learning algorithms was compared. Random forests achieved the best results for classification with equally weighted classes. When penalizing false positives more than false negatives, classification based on logistic regression achieved the best results. A recent study by Gansterer and Pölz [GP09] also focussed on feature-based phishing detection and aims at extending and refining existing methods. The authors proposed 15 new features which turned out to be crucial for the achieved classification accuracy. Their results show that the classification model based on a support vector machine (cf. Section 4.5) achieved the highest accuracy amongst all classifiers and that a ternary classification (ham vs. spam vs. phishing, cf. Section 5.1.1) worked better than a sequence of two binary classification steps for this problem.

Conceptually, feature-based email classification seems to be a promising approach for phishing detection, especially in combination with the ternary classification problem formulated above instead of the classical binary one. This allows for specifically targeting phishing messages in the enormous volume of regular and unsolicited email (spam). Open issues are the selection of relevant features and the combination of features, which are crucial tasks when dealing with large data in order to identify important features and decrease the computational cost for classifying unlabeled email. The goal is to identify features which are able to distinguish well between all three classes of email without a loss in classification accuracy. In Chapters 8 and 9, we investigate dimensionality reduction techniques based on NMF in order to address these open questions for ternary email classification problems.

5.2 Predictive QSAR Modeling

The increasing use of computational methods in the drug design and development process provides the possibility to analyze the interactions of small molecules with their target proteins on a quantitative level. This enables on the one side a deeper understanding of the molecular basis of drug-protein interactions, and on the other side a prediction of the biological activity of new molecules in a virtual screening run prior to their synthesis.

In the area of QSAR modeling (quantitative structure activity relationship modeling), quantitative representations of molecular structures are encoded in terms of information-preserving descriptor values. These descriptors are then used to determine the pharmacological or biological activity of molecular structures, which describes the beneficial or adverse effects of a drug in an organism. The big advantage of this process is the possibility to save time- and cost-expensive clinical trials by predicting the pharmaceutical activity of compounds with computational learning methods. Generally speaking, the pharmacological activity is a function of the properties of molecule structures, such that $activity = f$(physiochemical and/or structural properties). Compounds which have a particular pharmaceutical activity are also called *inhibitors*, *agonists* or *substrates* of a protein.

Chemical descriptors: In the literature (cf. [Ltd09]), a chemical descriptor is defined as the *final result of a logical and mathematical procedure which transforms chemical information encoded within a symbolic representation of a molecule into a useful number or the result of some standardized experiment.* In other words, chemical descriptors are used to derive and represent molecular and physicochemical properties of compounds, such as their 1D, 2D or 3D structure. In the context of data mining and machine learning, descriptors are also often referred to as *features* or *attributes.*

Similarity principle: Compounds having similar chemical structures (i. e., descriptor similarity) usually possess similar physicochemical properties and biological activities. Thus, compounds can be classified according to their chemical structures. Similar to email filtering, where unlabeled email is classified based on a set of already classified email samples (training set), unlabeled compounds can thus be classified based on a training set of compounds, whose pharmacological activity is known.

Descriptors are usually computed from structural codes, such as SMILES (simplified molecular input line entry specification (cf. [Wei88, WWW89]). SMILES is a simplified chemical notation that allows the representation of 2D chemical structures in linear textual form and supports also representation of some 3D characteristics, such as bold and hatched bonds, as well as charged and complexed molecular entities. According to [TC00], the most frequently used descriptors can be divided into 18 classes, the most important of them being *geometrical descriptors* such as volume and surface area, as well as *constitutional* and *topological descriptors* such as molecular weight, number of rings and rotatable bonds. A complete listing of these classes can also be found in [DJT08].

Chemical descriptors comprise different levels of complexity and thus the computational cost for determining them varies. Nowadays a variety of software tools is available to translate the textual representations of chemical structures into descriptor numbers. Among them, widely used tools for deriving molecular descriptors are DRAGON [TGT05], MOE [Gro09], Molconn-Z [Edu09], and JOELib [LYU07b].

The development of new chemical descriptors is still a topic of major interest in Chemoinformatics. Their choice as well as their number strongly influences the computational cost of steps 1-4 in Section 2.4. Especially the extraction of the data (step 1) can be a very costly process. Moreover, the data mining step (step 4) can be computationally very expensive if the number of features (descriptors) is large. Descriptors most suitable for representing compounds of a particular property can be selected either by intuition or, preferably, more systematically by means of feature selection or dimensionality reduction methods (cf. Chapters 3 and 6).

Application areas: The concept of avoiding drug-ligand interactions has raised high interest in recent years – especially in the field of ADMET, which refers to the absorption, distribution, metabolism, excretion, and toxicity properties of a molecule within an organism. Optimizing these properties during an early phase of the drug discovery process is crucial for reducing ADMET problems later in the development process. Almost one third of all compounds in the drug development process pipeline fail due to improper ADMET behavior which renders predictive ADMET an important issue in drug discovery (cf. [PVR03, TE08a]). Several key proteins identified so far in the ADMET cascade appear to show polyspecific interactions in the binding of their ligands. The goal of QSAR modeling is to identify these proteins, since they influence the pharmacokinetic or toxicological profile of a drug candidate. These proteins – so called *antitargets* – include nuclear hormone receptors (e. g., PXR and CAR), cytochrome P450 enzymes, several ABC transporters, and the hERG potassium channel.

- **Nuclear hormone receptors** (NHR) are a class of proteins that regulate gene expression by activating genes, and therefore leading to the up-regulation of various other proteins (such as ABC transporters and CYP enzymes).

- **Hepatic cytochrom-P450 enzymes** (e.g. CYP3A4, 2D6) are liver enzymes that are responsible for detoxification. Drug inhibition of CYP450 may lead to impaired liver function and toxic side effects.

- The members of the **ATP-binding cassette** (ABC) transporter family are membrane-bound efflux transporters, which can be divided into seven classes (ABCA-ABCG). The best studied class so far is the ABCB1 class, which is also called P-glycoprotein.

- The **human-ether-a-go-go-related gene** (hERG) channel, is a potassium-channel of the heart muscle. Drug inhibition of hERG leads to arrhythmia and can thus result in sudden death.

Although the proteins mentioned above are different with respect to their cellular localization, their three dimensional structure and also their biochemical function, they have one striking feature in common: they are polyspecific in recognizing their ligands, i. e., all of them interact with small molecules that are unrelated in their chemical structure as well as in their pharmacological function. This polyspecifity renders the identification of the "most useful" descriptors reflecting the relationship between the biological activity (discrete or continuous) and the calculated properties extremely difficult. The large number of available descriptors requires techniques for (*i*) identifying the "best" descriptors to discriminate between classes, and, (*ii*) reducing the dimensional space of the descriptors. There are several techniques to achieve one or both of these goals, but careful investigation of these methods is needed to avoid loss of information. In Part II of this thesis, several feature reduction (feature selection and/or dimensionality reduction) techniques applied on data coming from the QSAR modeling field are compared in terms of classification accuracy and runtime.

5.2.1 Classification Problem for QSAR Modeling

Classical QSAR modeling can either be defined as a regression problem or as a classification problem. In the problem setting of *binary* classification, every compound under investigation has to be classified as "active" or "inactive". In this context, "positives" are compounds that have a particular pharmaceutical activity such as inhibitors or substrates of a protein. On the contrary, "negatives" are compounds that do not have this particular pharmaceutical activity. Consequently, a true positive (TP) is an active compound which was correctly classified as active, and a false positive (FP) is an inactive compound which was wrongly classified as active. True negatives (TN) are thus inactive compounds which were correctly classified as inactive, and false negatives (FN) are active compounds which were wrongly classified as inactive.

In the field of QSAR modeling, the true positive rate is sometimes also referred to as *sensitivity*, *accuracy on actives* or *A1*, and the true negative rate is sometimes referred to as *specificity*, *accuracy on inactives* or *A0*.

5.2.2 Relevant Literature for QSAR Modeling

A wide variety of data mining and machine learning algorithms have been applied to establish in-silico models related to antitargets. A very recent and comprehensive collection of these efforts can be found in the book by Vaz and Klabunde [VK08]. Furthermore, a review of machine learning algorithms and their performance with respect to various ADMET related proteins is given in [LYU07a]. Several studies in the area of QSAR modeling have proven that feature reduction algorithms can lead to improvements in the classification performance and that the results achieved with reduced feature sets are generally easier to interpret. In the following, several studies related to molecular modeling focusing on feature reduction and classification are summarized briefly. The studies are grouped according to the type of antitargets under investigation. A survey of feature selection techniques for predictive QSAR modeling techniques for polyspecific drug targets can be found in [DJT08].

Ung *et al.* [ULY07] have presented a comprehensive in-silico prediction study that addresses the problem of discriminating between *PXR* activators and non-activators. They used a wrapper approach with recursive feature elimination (cf. Section 3) applied to three different learning methods (kNN, SVM and neural networks). Obtained results show that the classification performance based on reduced feature sets outperforms the classification results achieved with all descriptors by 3-5%.

Yap *et al.* [YC05] applied two distinct adaptations of a support vector machine (called PPCSVM – "positive probability consensus" and PMCSVM – "positive majority consensus", respectively) for the discrimination of substrates and inhibitors on three different datasets for three different *CYP450* enzymes. The chemical space of the datasets under investigation was initially represented by 1607 descriptors and a genetic algorithm wrapper was used to remove irrelevant descriptors. The interesting results show that the same descriptor classes were identified to be important for all three classification problems, i. e., on different CYP450 enzymes. This suggests that there are several descriptor classes suitable for describing any CYPP450 classification problem.

A large number of research studies focused on the problem of predicting *ABCB1* substrates and non-substrates. Xue *et al.* [XYS04, XSY04] described the classification performance of a support vector machine applied to three different ABCB1 datasets, and applied a recursive feature elimination (RFE) wrapper (cf. Section 3.2.2) as feature selection method. The authors observed a significant increase in the prediction accuracy when applying RFE (up to 11%) compared to a classification on the complete desriptor set. De Cerqueira-Lima and colleagues [CGO06] utilized a method called BinaryQSAR to construct ABCB1 substrate/non-substrate classification models. BinaryQSAR (developed by Labute (cf. [Lab99])) is a technique based on Bayesian classification (cf. Section 4.9) which is implemented in the MOE package (cf. [Gro09]) and can be applied on binary data only. Since this technique assumes that descriptors are uncorrelated to each other, a principle component analysis is usually performed prior to the classification step. Obviously, the number of principle components used within the classification process can be much smaller than the original dimensionality of the data (cf. Section 3.3.2). The results indicate that BinaryQSAR is able to achieve comparable and often better results than kNN, and is able to outperform a support vector machine classifier when using descriptors computed with the MOE package. Other work using feature reduction in combination with ABCB1 classification comprises the study by Svetnik *et al.* [SLT03], who evaluated the application of random forests (cf. Section 4.6) for the classification problem of ABCB1 substrates, the study by Huang *et al.* [HMM07], who established an SVM model for ABCB1 substrates which is optimized by a wrapper approach based on a particle swarm algorithm, or the study by Wang *et al.* [WTH05], who evaluated the performance of a self-organizing map by applying stepwise discriminant analysis (cf. Section 4.9) as feature selection method to the problem of ABCB1 classification.

Several studies focused on the application of feature reduction methods to *hERG* channel blockers. Seierstad *et al.* [SA06] compared principal component analysis (PCA, cf., Section 3.3.2) to randomized wrapper methods such as simulated annealing and deterministic wrapper methods using stepwise forward selection for a hERG ligand regression problem. It can be seen from their results that PCA performs comparable with most of the other methods, while simulated annealing achieves the best results for this problem setting. In a study by Thai and Ecker [TE08b], BinaryQSAR (explained above) was applied to the task of classifying hERG channel blockers and non-blockers. Their results show that in some cases models derived with fast and easy to calculate 2D molecular descriptors achieve similar results as models based on the complete feature set. In a later study [TE08a], the same authors classified hERG blockers based on a feature selection technique called QuaSAR-contingency (a method available in the MOE package) and a supervised adaption of a self-organizing map called counter-propagation network (cf. Section 4.9) as a learning algorithm. Their results show that counter-propagation neural networks with a 3-dimensional output layer combined with a set of 11 hERG relevant descriptors achieved the best performance. In another study by Tobita *et al.* [TNN05], support vector machines were used for both, feature reduction (as a deterministic wrapper approach) and classification. For two different test sets, their classification models achieved remarkable 90% and 95% accuracy, respectively. Other interesting work for hERG classification comprises the study by Roche *et al.* [RTZ02], who applied feature reduction based on self-organizing maps (SOMs, cf. Section 4.9) in combination with (supervised) artificial neural networks as classification process. Moreover, Dubus *et al.* [DIP06] used correlation-based feature selection (a representative of multivariate filter methods, cf. Section 3.2.1), and in the studies by Keserü *et al.* [Kes03] and Ekins *et al.* [EBS06], PCA was successfully applied to reduce the number of descriptors.

A different approach was studied by Hull *et al.* [HFS01, HSN01], who used latent semantic indexing for structure indexing. In their LaSSI (latent semantic structure indexing) approach, they used LSI for similarity searching based on chemical substructure descriptors. The underlying rationale of using this method is that LSI might represent an useful alternative to determine similarity on different descriptor sets. Compared to an "in-house" similarity method (which uses the Dice similarity definition (cf. [Rij79])) called TOPOSIM (cf. [HFS01]), LASSI achieves comparable and sometimes slightly better results. Moreover, the results highlight that LaSSI selects very different compounds than TOPOSIM.

5.3 Comparison of Areas

The data coming from these two different application areas differ strongly in several aspects, such as the number of available training samples, the type and quality of features, or the general characteristics (sparsity, range, type, . . .) of the data.

Concerning the *number of labeled training samples*, there is a big difference between the two areas mentioned before. Since labeled and unlabeled email is rather easy to collect (for example, from TREC [NIS09] or Enron [Coh09]) the number of training and test samples can easily reach many thousands (for all three groups, ham, spam and phishing). Contrary to that, the number of available labeled compounds for prediction tasks related to QSAR modeling is often relatively small (usually a few hundred compounds). This is due to the fact, that the number of compounds that have already undergone time- and cost-expensive clinical trials is limited. Another possibility to gather pre-classified compounds is to collect them from the literature (such as the national cancer institute). Unclassified compounds can be extracted much easier, for example, from compound libraries such as SPECS [SPE09] or ChemDiv [Che09]. The limited number of labeled training samples is a big challenge and highlights the importance of accurate sampling strategies and techniques such as cross-validation to avoid the problem of overfitting (cf. Section 2.4).

Comparing the *features* of data coming from email filtering and data coming from QSAR modeling also shows several differences for both application areas. Most post-send email filtering methods rely on well established feature sets comprising features extracted from the (textual) content of the email message or on special properties of the email (such as date, size, number of html tags, etc.). Most of these features can be extracted very fast, for example, with rule-based filtering systems such as SpamAssassin (cf. Section 5.1.2). The fast extraction of features is very important in this context, since the time available for classifying a single email message is strongly limited, especially in high-load situations on the receiving SMTP server. Chemical descriptors, on the contrary, comprise different levels of complexity and thus the computational cost for determining them often varies significantly. Since the same descriptors are often used to predict the pharmacological activity of compounds from different groups of antitargets (hERG, CYP, etc.), the extraction of the ideal feature set is usually more difficult than in the context of email filtering.

Feature sets used in the context of email filtering are usually well established and thus the choice of good feature sets is easier than in drug discovery applications. To give an example, the SpamAssassin system utilizes a built-in pseudo feature ranking based on the values of each rule/test (determined using a neural network). Contrary to that, feature sets used in the drug discovery context are often not so well established and thus the choice of the "best" feature set (in terms of classification accuracy) is a more difficult task than for email filtering. This issue is investigated in detail in Chapter 6 of this thesis.

Moreover, there is a big difference in the *general characteristics* of the data from both application areas. Data coming from the context of email filtering are usually much sparser than data coming from QSAR modeling. For example, usually only a small number of tests integrated in rule-based email filtering systems triggers for an email message (cf. Section 5.1.2). When considering purely text-based features, the resulting term $\times$ document matrices are usually also very sparse (cf. Section 4.7). Moreover there is a big difference in the range and the type (nominal, numeric, discrete, etc.) of attributes.

Due to these differences in the data and the application areas the typical accuracy achieved with classification algorithms differs strongly. Spam filtering systems often achieve high classification overall accuracy of 95% or more. The accuracy achieved with data coming from the drug discovery field differs strongly and is usually much lower. Depending on the data, the achieved classification accuracy may range from 60% up to 95% or more, but it is usually somewhere in between.

In Part II of this thesis, several new approaches and new algorithmic variants of feature reduction and classification techniques will be investigated and applied to datasets coming from the application areas of email filtering and drug discovery.

Part II

New Approaches in Feature Reduction and Classification

Chapter 6

On the Relationship Between Feature Reduction and Classification Accuracy

6.1 Overview of Chapter

In this chapter, we investigate the relationship between several attribute space reduction techniques and the resulting classification accuracy for the two application areas mentioned in Chapter 5. On the one hand, we consider email filtering, where the feature space contains various properties of email messages, and on the other hand, we consider drug discovery problems, where quantitative representations of molecular structures are encoded in terms of information-preserving descriptor values. Subsets of the original attributes constructed by feature selection techniques (such as filter and wrapper methods, cf. Section 3.2) as well as subsets of linear combinations of the original attributes constructed by three different variants of the principle component analysis (PCA, cf. Section 3.3.2) are compared in terms of the classification performance achieved with various machine learning algorithms as well as in terms of runtime performance. We successively reduce the size of the attribute sets and investigate the changes in the classification results. Moreover, we explore the relationship between the variance captured in the linear combinations within PCA and the resulting classification accuracy.

Results show that the classification accuracy based on PCA is highly sensitive to the type of data and that the variance captured by the PCs is not necessarily a vital indicator for the classification performance.

6.2 Introduction and Related Work

As the dimensionality of the data increases, many types of data analysis and classification problems become significantly harder. The reasons for this phenomenon – often referred to as *curse of dimensionality* – are outlined in Chapter 3 together with a discussion of feature selection (FS) and dimensionality reduction (DR) methods. Several extensive surveys of various feature selection and dimensionality reduction approaches can be found in the literature, for example, in [MBN02] or [GE03].

There are several studies that have investigated the classification accuracy of machine learning algorithms based on feature sets extracted by PCA. [HMO06] have investigated the effect of PCA on machine learning accuracy (for C4.5, RIPPER, linear SVM, RBF SVM, kNN and linear regression) with high-dimensional data based on different pre-processing steps. They use the NIPALS method [GK86] to iteratively compute only the first n principle components (PCs) of a data sample until a required number of PCs have been generated. Their results show that using PCA in combination with classification may improve the classification accuracy when dealing with high dimensional data. For carefully selected pre-processing techniques, the authors show that using principal components instead of the original features results in either the same accuracy (i. e., same error, for a C4.5 and a RIPPER classifier) or a numerically higher accuracy (i. e., smaller error, for linear SVM, RBF SVM, kNN and linear regression). [Pop01] has analyzed the effect of PCA on three different machine learning methods (C5.0, instance-based learner and naïve Bayes). In one test-run, the first n PCs (i. e., linear combinations of the original attributes) were added to the original attributes, in the second test run, the principle components replaced them. The results show that adding the PCs to the original attributes slightly improved the classification accuracy for all machine learning algorithms (mostly for small numbers of n), whereas replacing the original attributes only increased the accuracy for one algorithm (naïve Bayes).

6.3 Open Issues and Own Contributions

Different techniques can be applied to perform a principle component analysis, for example, the eigenvalue decomposition of either the covariance or the correlation matrix can be used (cf. Section 3.3.2). Another question is how the scaling of the original data affects the PCs and the resulting classification accuracy. Attributes resulting from these different PCA variants differ significantly in their coverage of the variability of the original attributes. To the best of our knowledge no systematic studies have been carried out to explore the relationship between the variability captured in different variants of the PCA and the accuracy of machine learning algorithms operating on them. One of the objectives of this chapter is to summarize investigations of this issue. More generally, we investigate the variation of the achieved classification accuracy depending on the choice of specific variants for calculating the PCA for two very different datasets. Other important aspects motivating such investigations are questions relating to how the classification accuracy based on PCA subsets compares to classification accuracy based on subsets of the original features of the same size, or how to identify the smallest subset of original features which yields a classification accuracy comparable to the one of a given PCA subset.

In the following, several feature sets (and subsets of these feature sets) are compared in terms of resulting classification accuracy. The three types of feature subsets considered are based on feature ranking (using the filter method information gain, cf. Section 3.2.1), a wrapper subset evaluator (cf. Section 3.2.2) and three different variants of PCA.

6.4 Datasets

The datasets used for experiments come from the two different application areas mentioned in Chapter 5 and differ strongly in the number of instances and features and in their characteristics.

Email data. The first dataset consists of 10 000 email messages (half of them spam, half not spam) taken from the TREC 2005 email corpus [CL05]. The values of the features for each message were extracted using SpamAssassin [Spa09] (cf. Section 5.1.2), where different parts of each email message are checked by various tests, and each test assigns a certain value to each feature (positive feature values indicate spam messages, negative values indicate non-spam messages). Although the number of features determined by SpamAssassin is rather large, only a relatively small number of these features provide useful information. For the dataset used only 230 out of more than 800 tests triggered at least once, resulting in a $10\,000 \times 230$ data matrix.

Drug discovery data. The second dataset comes from medicinal chemistry (cf. Section 5.2). This dataset consists of 249 structurally diverse chemical compounds. 110 of them are known to be substrates of P-glycoprotein, a macromolecule which is notorious for its potential to decrease the efficacy of drugs ("antitarget"). The remaining 139 compounds are non-substrates. The chemical structures of these compounds are encoded in terms of 366 information preserving descriptor values (features). Hence, our drug discovery (DD) dataset is a 249×366 matrix.

Data characteristics. Whereas the email dataset is very sparse (97.5% of all entries are zero) the drug discovery dataset contains only about 18% zero entries. Moreover, most of the email features have the property that they are either zero or have a fixed value (depending on whether a test triggers or not). This is completely different from the drug discovery dataset where the attribute values vary a lot (the range can vary from descriptors represented by small discrete numbers to descriptors given by floating-point values out of a theroretically contiuous range).

6.5 Feature Subsets

For both test datasets we determined different feature subsets, i. e., out of the original data matrix D we computed a new matrix $D^{'}$. For the FS subsets (information gain and wrapper), D and $D^{'}$ differ only in the number of columns (attributes), for the PCA subsets they differ in the number of columns *and* in their interpretation (i. e., the columns are linear combinations of the original attributes, cf. Section 3.3).

For extracting the *wrapper subsets* we used Weka's (cf. [WF05]) *wrapper subset evaluator* in combination with the *best first* search method. A paired t-test was used to compute the probability if other subsets may perform substantially better (cf. Section 3.2.2). For the investigations in this chapter, neither a maximum nor a minimum number of features was pre-defined. The optimum number of features was automatically determined within the wrapper subset evaluator (based on the t-test). For our evaluations, between 4 and 18 features were selected.

As a *filter approach* we ranked the attributes with respect to their information gain (IG). As mentioned in Section 3.2, this ranking is independent of a specific learning algorithm and contains – before selecting a subset – all attributes (ranked in desending order with respect to their IG).

For *dimensionality reduction*, we studied PCA. In the literature (cf., for example, [Jol02, TSK05, WF05, Mat09b]), several variants appear. We investigated the differences of three such variants (denoted by PCA_1, PCA_2 and PCA_3) in terms of the resulting classification accuracy. In all cases we first performed a mean shift of all features such that the mean for each feature becomes 0. We denote the resulting feature-instance matrix by M. Based on this first preprocessing step we define three variants of the PCA computation. The resulting datasets contain min(features, instances) linear combinations of the original attributes (out of which a subset is selected).

PCA_1: The eigenvalues and eigenvectors of the covariance matrix of M (cf. Section 3.3) are computed. The new attribute values are then computed by multiplying M with the eigenvectors of the covariance matrix $\text{Cov}(M)$.

PCA$_2$: The eigenvalues and eigenvectors of the matrix containing the correlation coefficients of M (cf. Section 3.3) are computed. The new attribute values are then computed by multiplying M with the eigenvectors of the correlation coefficient matrix Corr(M).

PCA$_3$: Each feature of M is first normalized by its standard deviation (i. e., z-scored). These normalized values are then used as input for the eigenvalue decomposition (i. e., there is no difference between the correlation coefficient matrix Corr(M) the and covariance matrix Cov(M)) and also for the computation of the new attributes.

6.6 Machine Learning Methods

For evaluating the classification performance of the reduced feature sets we used six different machine learning methods (see Chapter 4 for a description of these methods). Experiments were performed with a support vector machine (SVM) based on the sequential minimal optimization algorithm using a polynomial kernel with an exponent of 1; a k-nearest neighbor (kNN) classifier using different values of k (1 to 9); a bagging ensemble learner (Bagging) using a pruned decision tree as base learner; a single J.48 decision tree (J.48) based on Quinlan's C4.5 algorithm; a random forest (RandF); and a Java implementation of the RIPPER rule-based learner (JRip).

6.7 Experimental Results

For all feature sets except the wrapper subsets we measured the classification performance for subsets consisting of the n "best ranked" features (n varies between 1 and 100). For the information gain method, the top n information gain ranked original features were used. For the PCA subsets, the first n PCs capturing most of the variability of the original attributes were used. The classification accuracy was determined using a 10-fold cross-validation. The results are shown separately for the two datasets described in Section 6.4. For kNN, only the kNN(1) results are shown since $k = 1$ mostly yielded the best results. For comparison we also classified completely new data (separating feature reduction and classification process). In this setting, we performed the feature selection step (using information gain and a wrapper approach) on a separate dataset which was then not used for building or testing the classification model. In most cases the accuracy was similar.

For the experimental evaluation we used Matlab [Mat09b] for computing three different variants of the PCA (cf. Section 6.5), and the Weka toolkit [WF05] for computing the feature selection subsets (information gain and wrapper approach) and for measuring the classification performance of the learning methods on each of these feature sets.

6.7.1 Email Data

Table 6.1 shows the overall classification accuracy for the information gain subsets and the PCA subsets. The results are shown as the *average* results over all subsets of the top n features (for IG) and PCs (for PCA), respectively, for $n = 1, \dots, 100$. Besides, the average classification accuracy over all algorithms is shown (AVG.). The best and the worst average results for each learning algorithm over the feature reduction methods are highlighted in bold and italic letters, respectively. The best overall result over all feature sets and all learning algorithms is marked with an asterisk. Since the wrapper subsets contain a fixed number of features, the wrapper results are shown in Table 6.2 only.

Table 6.1: Email data – average overall classification accuracy (in %).

	SVM	kNN(1)	Bagging	J.48	RandF	JRip	AVG.
Infogain	*99.20*	*99.18*	*99.16*	*99.16*	*99.21*	*99.18*	*99.18*
PCA$_1$	99.26	**99.70**	**99.68**	**99.70**	* **99.77**	99.67	99.63
PCA$_2$	**99.55**	99.69	99.67	99.69	99.75	**99.68**	**99.67**
PCA$_3$	99.54	99.64	99.65	99.64	99.65	99.64	99.63

Table 6.2 shows the *best* classification accuracy for all feature subsets (including the wrapper subsets) and for a classification based on the complete feature set. Table 6.2 also contains the information how many original features (for FS) and how many principal components (for PCA) were needed to achieve the respective accuracy. Again, the best and worst results are highlighted.

Table 6.2: Email data – best overall classification accuracy (in %).

	SVM	kNN(1)	Bagging	J.48	RandF	JRip
All features	99.75 *230 attr.*	99.70 *230 attr.*	99.71 *230 attr.*	99.65 *230 attr.*	99.73 *230 attr.*	99.66 *230 attr.*
Wrapper *fixed set*	*99.61* *7 attr.*	*99.60* *5 attr.*	*99.61* *4 attr.*	*99.61* *7 attr.*	*99.67* *11 attr.*	*99.64* *7 attr.*
Infogain	**99.76** ***100 attr.***	99.71 *50 attr.*	99.70 *50 attr.*	99.71 *100 attr.*	99.78 *80 attr.*	99.72 *90 attr.*
PCA$_1$	*99.65* *90 PCs*	99.74 *40 PCs*	99.69 *40 PCs*	99.75 *30 PCs*	* **99.82** ***70 PCs***	99.73 *5 PCs*
PCA$_2$	99.67 *90 PCs*	**99.75** ***40 PCs***	**99.72** ***30 PCs***	**99.78** ***60 PCs***	99.80 *15 PCs*	**99.76** ***40 PCs***
PCA$_3$	99.65 *100 PCs*	99.73 *5 PCs*	99.71 *20 PCs*	99.71 *4 PCs*	99.79 *15 PCs*	99.73 *50 PCs*

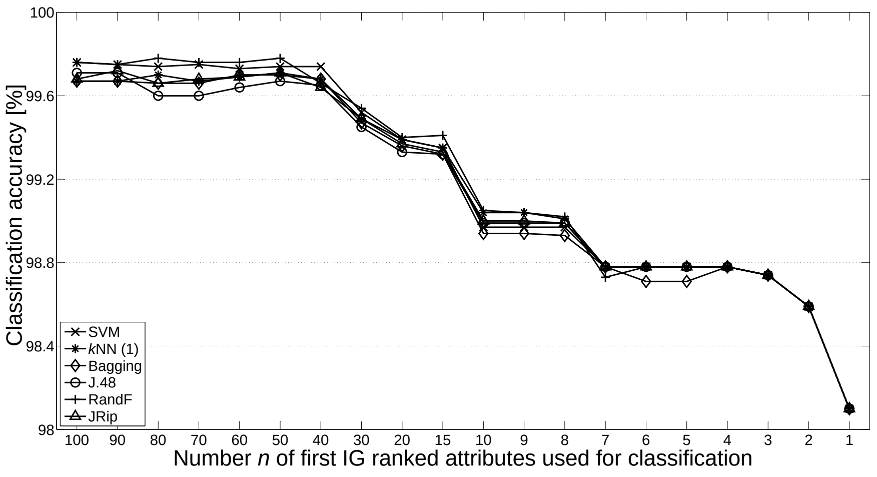

Figure 6.1: Email data – information gain subsets.

Feature subset selection. A comparison of the two feature selection methods shows that the best accuracies achieved with IG subsets are better than the wrapper results (see Table 6.2). Nevertheless, when looking at the size of the subsets with the best accuracy it can be seen that the wrapper subsets are very small. Figure 6.1 shows the degradation in the classification accuracy when the number of features in the IG subsets is reduced. Interestingly, all machine learning methods show the same behavior without any significant differences. The accuracy is quite stable until the subsets are reduced to 40 or less features, then the overall classification accuracy tends to decrease proportionally to the reduction of features. Comparing the wrapper results with the IG results with the same number of attributes (4 to 11 features, see Figure 6.1 and Table 6.2), it can be seen that the wrapper approach clearly outperforms IG. Moreover, the classification accuracy achieved with the wrapper subsets is only slightly worse than the accuracy achieved with the complete feature set, but with a reduction of about 96% of the feature space.

PCA. Figures 6.2, 6.3 and 6.4 show the overall classification accuracies for the three PCA variants. Generally, the results are very stable regardless of the number of principle components used for classification. Surprisingly, only the accuracy of the SVM method (using a polynomial kernel) clearly decreases with a smaller number of principal components used, especially so for PCA$_1$, but also for PCA$_2$ and slightly for PCA$_3$.

Explaining the variance. Even though the classification results for all three PCA variants are similar, it is very interesting to note that when the correlation matrix is used instead of the covariance matrix to compute the eigenvectors and eigenvalues (as it is the case for PCA$_2$ and PCA$_3$) the fraction of the variance captured by the

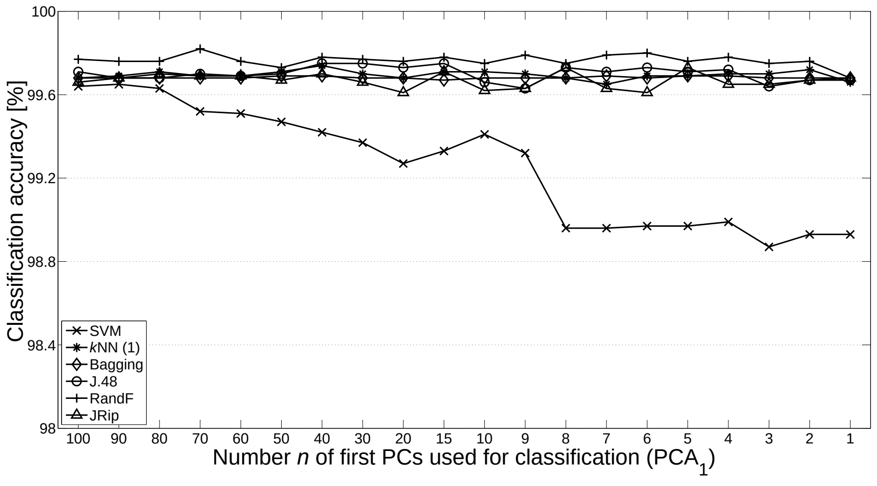

Figure 6.2: Email data – PCA$_1$ subsets.

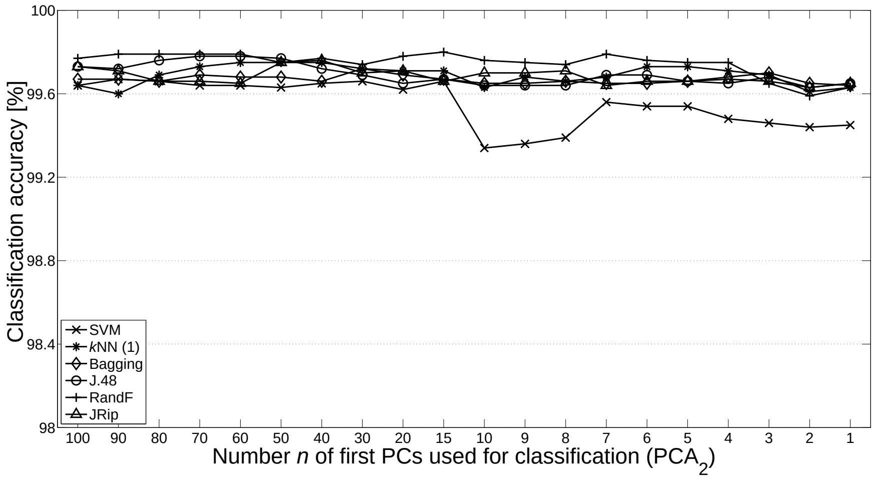

Figure 6.3: Email data – PCA$_2$ subsets.

first n PCs (i. e., the accumulated percentage of the first n eigenvalues) decreases remarkably (see Table 6.3).

Algorithms. Although the overall classification accuracy in Figures 6.1, 6.2, 6.3, and 6.4 is very good in general, when comparing the different machine learning methods it can be seen that random forest achieves the best results for all of the reduced feature sets used, and that the SVM classifier seems to be most sensitive to the size of the PC subsets. For the average PCA results (shown in Table 6.1 and Figure 6.2), SVM shows the lowest classification accuracy. When the complete feature set is used, the SVM results are slightly better than the random forest results (Table 6.2).

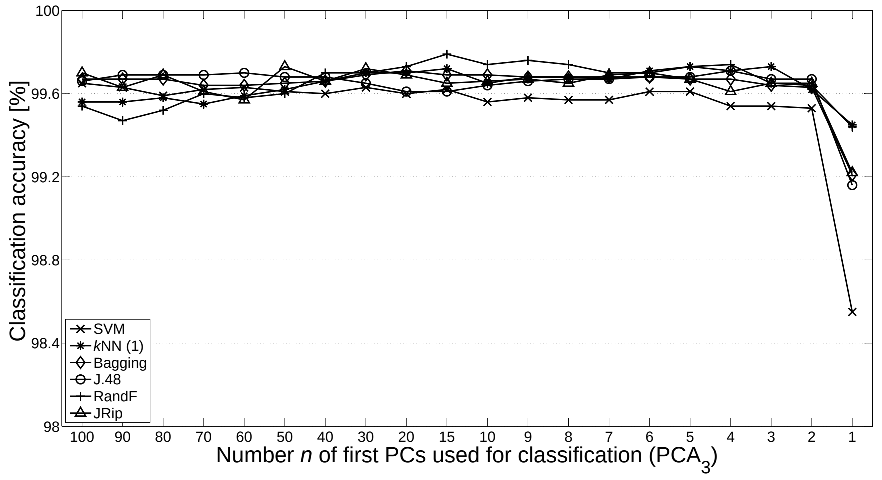

Figure 6.4: Email data – PCA_3 subsets.

Table 6.3: Email data – % variance captured by first n PCs (max. dim: 230).

n	100	80	60	40	20	10	5	4	3	2	1	
PCA_1	99.4	98.9	97.8	95.3	87.0	75.7	62.7	58.9	54.0	48.0	38.0	%
$PCA_{2,3}$	67.7	58.9	49.4	38.3	24.7	15.8	10.3	9.0	7.7	6.3	3.9	%

6.7.2 Drug Discovery Data

Tables 6.4 and 6.5 show average and best classification results over different feature subsets for the drug discovery data.

Table 6.4: Drug discovery data – average overall classification accuracy (in %).

	SVM	*k*NN(1)	Bagging	J.48	RandF	JRip	AVG.
Infogain	**69.25**	**67.55**	70.63	**68.47**	68.04	**70.32**	**69.04**
PCA_1	63.50	*65.87*	*65.84*	*61.81*	*65.03*	*65.74*	*64.63*
PCA_2	*61.05*	66.39	69.27	65.27	67.16	65.92	65.84
PCA_3	68.78	67.28	* **71.02**	63.76	**69.66**	67.06	67.93

Feature subset selection. A comparison of Tables 6.4 and 6.5 (drug discovery dataset results) with Tables 6.1 and 6.2 (email dataset results) shows that the classification accuracy achieved with the drug discovery dataset is generally much lower than for the email dataset. Moreover, for the drug dataset, the average classification results (Table 6.4) achieved with IG tend to be much better compared to the PCA results than for the email dataset. For four out of the six learning algorithms used, the best average results were achieved with the IG feature set (cf. Table 6.4) – for the email dataset, the best average results were all achieved with PCA_1 or PCA_2 subsets (cf. Table 6.1).

Table 6.5: Drug discovery data – best overall classification accuracy (in %).

	SVM	kNN(1)	Bagging	J.48	RandF	JRip
All features	70.67 *367 attr.*	73.89 *367 attr.*	74.30 *367 attr.*	*64.24* *367 attr.*	73.52 *367 attr.*	69.09 *367 attr.*
Wrapper *fixed set*	**77.48** ***18 attr.***	**79.91** ***6 attr.***	**79.51** ***10 attr.***	**79.53** ***6 attr.***	* **79.93** ***6 attr.***	**79.89** ***6 attr.***
Infogain	72.70 *60 attr.*	73.08 *80 attr.*	74.31 *20 attr.*	71.11 *1 attr.*	71.89 *7 attr.*	73.52 *2 attr.*
PCA$_1$	70.69 *60 PCs*	*73.07* *15 PCs*	*68.68* *15 PCs*	65.87 *15 PCs*	*69.48* *4 PCs*	69.09 *6 PCs*
PCA$_2$	*65.89* *60 PCs*	71.89 *60 PCs*	73.08 *15 PCs*	68.29 *60 PCs*	73.89 *40 PCs*	*68.69* *4 PCs*
PCA$_3$	73.89 *6 PCs*	73.09 *10 PCs*	75.90 *6 PCs*	69.09 *5 PCs*	76.69 *10 PCs*	71.48 *7 PCs*

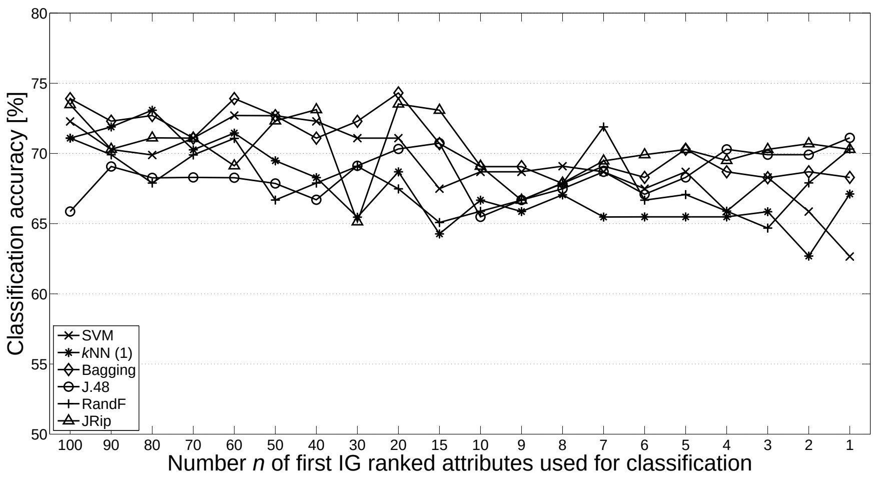

Figure 6.5: Drug discovery data – information gain subsets.

When looking at the best overall classification accuracy (Table 6.5), a very interesting observation is that the (again very small) wrapper subsets clearly outperform the complete feature set using all attributes and the IG subsets. In contrast to the results for the email data, this is also true for larger IG subsets with about 30-100 features (see Table 6.5). For the email dataset, the best overall classification accuracy (cf. Table 6.2) based on the wrapper subsets was always lower than the best overall classification accuracy based on IG or different PCA subsets. For the DD dataset, the three best wrapper results (achieved with kNN, random forest and JRip) were all achieved with a feature set containing only 6 attributes. Interestingly, only two of these attributes appear in all three subsets, the other attributes differ throughout the feature subsets.

Figure 6.5 shows the classification performance for the IG subsets with different sizes. There is a clear difference between this curve and the curve shown in Figure 6.1 (IG subsets for the email data). For the DD datasets, there is no proportional decline in the classification accuracy compared to the size of subsets used, as it is the case for the email datasets. For most small IG subsets the classification performance remains acceptable compared to large IG subsets, for the J.48 the results achieved with small IG subsets are even better than the results achieved with large subsets (cf. Figure 6.5).

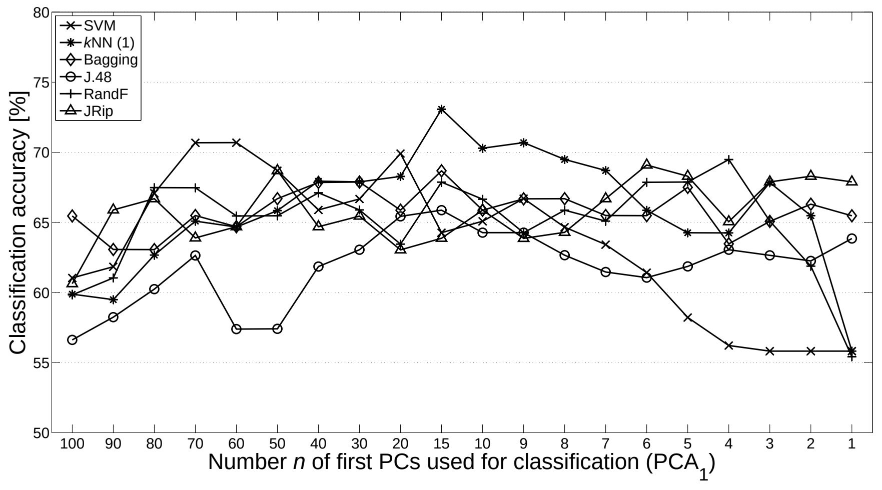

Figure 6.6: Drug discovery data – PCA_1 subsets.

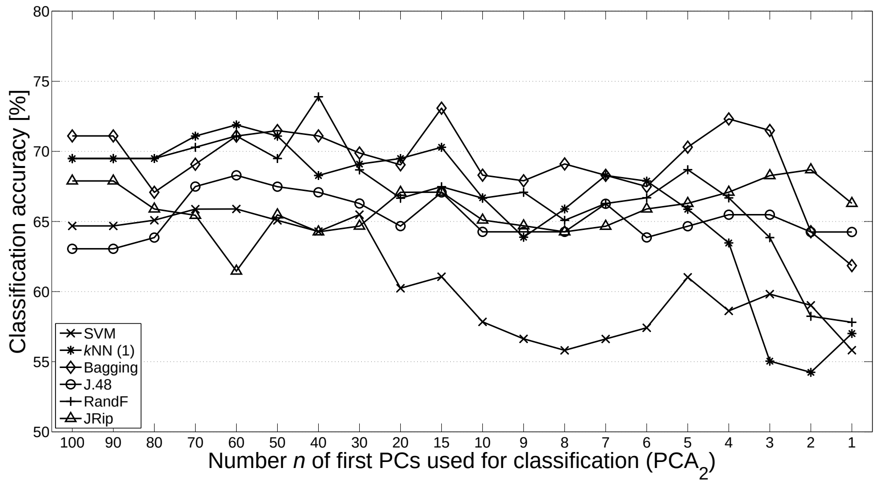

Figure 6.7: Drug discovery data – PCA_2 subsets.

PCA. Figures 6.6, 6.7 and 6.8 show the overall classification accuracies for the three PCA variants. The results are again very different from the results with the email dataset. Surprisingly, the accuracy between the first 90 to 100 PCs, as well as the results using only very few PCs tend to be much lower than the results for other PC numbers when using feature set PCA_1, and partly also when using feature set PCA_3.

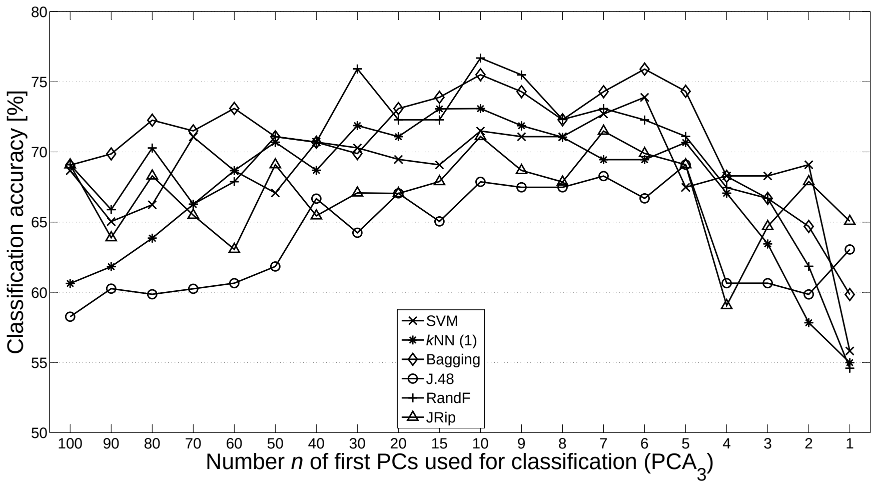

Figure 6.8: Drug discovery data – PCA$_3$ subsets.

For PCA$_2$ there is no such behavior. When comparing PCA$_1$, PCA$_2$ and PCA$_3$ (see also Tables 6.4 and 6.5), it can be seen that the results for some classifiers change significantly. For example, the best classification accuracy for SVM decreases by 5% even though both subsets achieved the best result with the same number of PCs (60, see Table 6.5). On the other hand, for bagging and RF the accuracy improved by about 4%. The best bagging results were again achieved with the same number of PCs (15). Comparing the average results (cf. Table 6.4) and the best results (cf. Table 6.5) of the PCA variants, it can be seen that the best PCA variant for the DD dataset is PCA$_3$.

Explaining the variance. The percentage of variance captured by the first n principal components is much higher than for the email dataset (cf. Table 6.6). This is due to the differences in size and characteristics of the datasets (see Section 6.4).

Table 6.6: DD data – % variance captured by first n PCs (max. dim: 249).

n	100	80	60	40	20	10	5	4	3	2	1	
PCA$_1$	100	99.9	99.9	99.9	99.9	99.8	98.7	98.1	97.0	94.6	87.9	%
PCA$_{2,3}$	99.6	99.1	97.9	95.2	88.7	79.9	68.2	63.6	58.1	51.4	40.6	%

Algorithms. Compared to the email dataset, the overall classification accuracy is much lower for all machine learning methods used. Comparing the six different algorithms, it can be seen that bagging achieves the best accuracy on the average results (Table 6.4). The best results for a given feature subset (Table 6.5) are either achieved with kNN, bagging or RF.

6.7.3 Runtimes

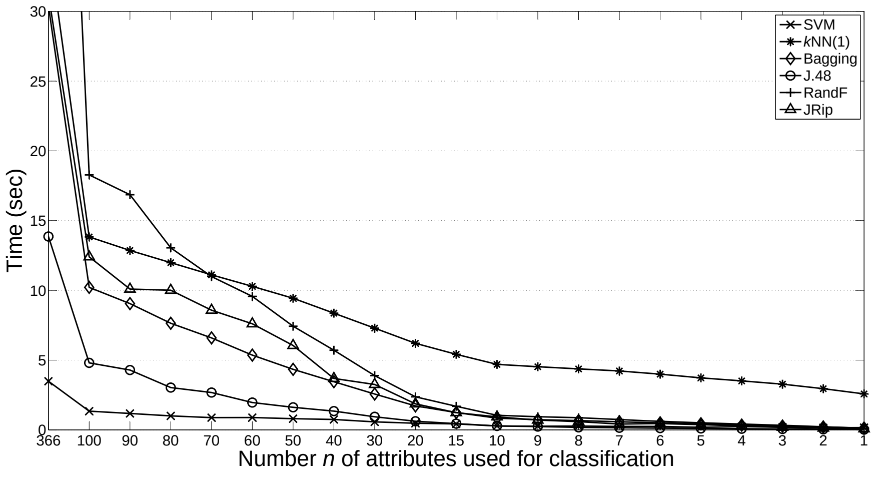

Figure 6.9: Email data – classification runtimes.

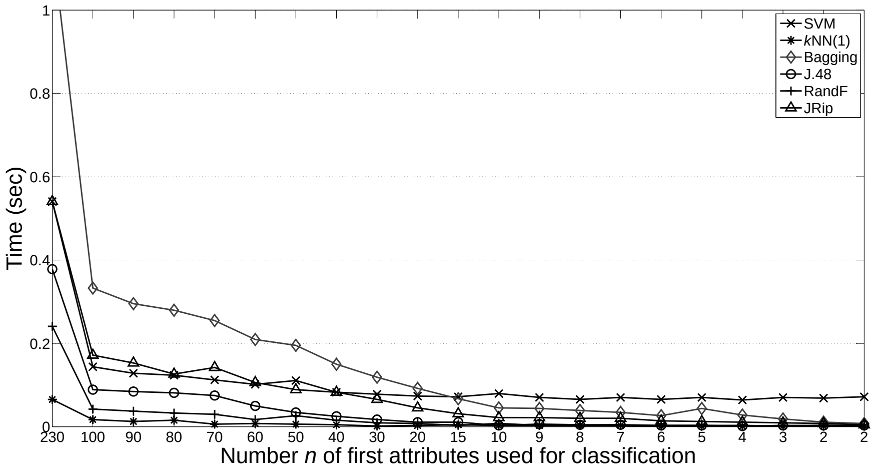

Figure 6.10: Drug discovery data – classification runtimes.

Figures 6.9 and 6.10 show the runtimes for the classification process (training *and* testing) for the email and the drug discovery dataset, respectively, for a ten-fold cross-validation using information gain subsets with different numbers of features and for the complete feature set. Obviously, the time needed for the classification process decreases with fewer attributes. For all machine learning algorithms except kNN, a big amount of the classification time is spent in the training step. Once the model has been built, the classification (testing) of new instances can be performed rather quickly. As kNN does not build a model but computes all distances in the classification process, there is no training step for this approach. Comparing the classification times, it can be seen that kNN is the slowest classifier on the larger email dataset but the fastest on the smaller drug discovery dataset. The fastest classifier on the email dataset is the support vector classifier (SVM), the slowest classifier on the drug discovery dataset is the bagging ensemble method.

Feature reduction runtimes. Table 6.7 shows the runtimes for different feature reduction processes, performed with Weka on the complete datasets. It is very interesting to note the big gap between the time needed to compute the feature ranking (IG) and the various wrapper methods. On the smaller drug discovery dataset, the slowest wrapper (WraRF, random forest) needed more than 48 minutes, on the email dataset more than 12 hours ! On the email dataset, the slowest wrapper (Wra*k*NN) needed even more than 20 hours, but was the fastest wrapper method on the drug discovery dataset. For a fair comparison, we used Weka's PCA routine (which is not the fastest available routine) for computing the PCs. The computation of the PCs is more time consuming than the computation of the information gain ranking, but much faster than the wrapper methods. A general discussion of feature ranking, wrapper, and dimensionality reduction methods in terms of runtime can be found in Chapter 3.

Table 6.7: Feature reduction runtimes (in seconds).

	IG	PCA	WraSVM	WraJ48	WraBag	WraRip	WraRF	Wra*k*NN
Email	5.1 e+0	8.1 e+1	3.0 e+3	4.4 e+3	7.2 e+3	1.2 e+4	4.4 e+4	7.4 e+4
DD	2.0 e−1	3.0 e+0	1.8 e+3	1.3 e+3	2.2 e+3	8.1 e+2	2.9 e+3	4.1 e+2

6.8 Discussion

We have investigated the relationship between various methods for feature reduction (feature subset selection as well as dimensionality reduction) and the resulting classification performance. More specifically, feature subsets determined with a wrapper method and information gain were compared to feature subsets of linear combinations of the original features, computed with three variants of the principle component analysis (PCA). Extensive experiments performed with datasets from two very different application contexts, email classification and drug discovery, lead to the following conclusions.

Datasets: When looking specifically at the two datasets investigated in this chapter, we note that the classification accuracy achieved with different feature reduction strategies is highly sensitive to the type of data. For the email data with quite well established feature sets there is much less variation in the classification accuracy than for the drug discovery data.

Feature reduction: Among the feature selection methods, wrapper methods tend to produce the smallest feature subsets with very competitive classification accuracy – in many cases they are the best overall feature reduction methods. Wrapper methods clearly outperform feature ranking methods based on IG on the drug discovery data, and also show acceptable classification accuracy on the email dataset. Although for the email data the best overall IG subset achieves better results than the wrapper subsets, the wrapper subsets lead to better accuracy than IG subsets with comparable sizes. However, wrapper methods tend to be much more expensive computationally than the other feature reduction methods.

For dimensionality reduction based on PCA, it is important to note that the three variants considered in this chapter tend to differ significantly in the percentage of the variance captured by a fixed number of principal components, in the resulting classification accuracy, and particularly also in the number of principal components needed for achieving a certain accuracy. It has also been illustrated clearly that the percentage of the total variability of the data captured by the principal components used is *not necessarily* correlated with the resulting classification accuracy.

Classification algorithms: An investigation of the applied machine learning algorithms shows that the SVM accuracy was surprisingly low on the PCA subsets. Even though SVMs perform very well when all features or subsets of the original features are used, they achieve only the lowest accuracy for all three PCA subsets on the email data. On the drug discovery data, SVMs achieve a reasonable accuracy only with a PCA_3 subset. The accuracy of most classifiers tends to be much less sensitive to the number of features when principal components are used instead of subsets of the original features, especially so for the email data. This is not surprising, since the principal components in general contain information from *all* original features. However, it is interesting to note that on the email data the SVM classifier is an exception: Its accuracy decreases clearly when fewer PCs are used, similar to the situation when feature subsets are used.

More generally speaking, the experimental results underline the importance of a feature reduction process. In many cases, in particular in application contexts where the search for the best feature set is still an active research topic (such as in the drug discovery application), the classification accuracy achieved with reduced feature sets is often significantly *better* than with the full feature set. In application contexts where good feature sets are already well established the differences between different feature reduction strategies are much smaller.

Chapter 7

Email Filtering Based on Latent Semantic Indexing

7.1 Overview of Chapter

In this chapter, the application of latent semantic indexing (LSI, cf. Section 4.8) to the task of detecting and filtering spam email is studied. Comparisons to the basic vector space model (VSM, cf. Section 4.7) and to the extremely widespread, de-facto standard for spam filtering, the SpamAssassin system (cf. Section 5.1.2), are summarized. It is shown that both VSM and LSI achieve significantly better classification results than SpamAssassin.

Obviously, the classification performance achieved in this special application context strongly depends on the feature sets used. Consequently, the various classification methods are also compared using two different feature sets: (*i*) a set of purely textual features of email messages that are based on standard word- and token extraction techniques, and (*ii*) a set of application-specific "meta features" of email messages as extracted by the SpamAssassin system. It is illustrated that the latter tends to achieve consistently better classification results. A third central aspect discussed in this chapter is the issue of problem reduction in order to reduce the computational effort for classification, which is of particular importance in the context of time-critical on-line spam filtering (for example, in combination with the greylisting technique mentioned in Section 5.1.2). In comparison to Chapter 6, where several explicit feature reduction methods are compared, the feature reduction step in this chapter is mostly performed implicitly within the LSI process. In particular, the effects of truncation of the singular value decomposition (SVD, see Section 3.3.3) in LSI and of a reduction of the underlying feature set (feature selection) are investigated and compared.

7.2 Introduction and Related Work

Spam email tends to have several semantic elements in common, which are usually not present in regular email. This assumption is plausible due to the economic aspects underlying the spam phenomenon (see, for example, [ISG06] for a discussion). However, the common semantic elements are hard to pinpoint comprehensively, because they are not fully known, not always explicit, and may change dynamically, etc. In other words, they are in some sense implicit or *latent*. Some previous approaches tried to concentrate on specific properties of (current) spam and to design anti-spam methods based thereon. However, due to the difficulties in identifying a comprehensive set of properties which unambiguously characterize spam, these approaches did not yet lead to fundamental and persistent anti-spam strategies. The application of several state-of-the-art information retrieval and text-mining techniques to the tasks of spam filtering and email classification is topic of current research (cf. related work for email filtering as summarized in Section 5.1.2).

An approach closely related to the methods investigated in this chapter has been proposed by Gee [Gee03], where an LSI-based classifier is used for spam filtering. Gee's results show that using LSI as the basis for an email classifier enjoys a very high degree of recall as well as a high degree of precision (cf. Section 2.4.5), while other classification methods such as Naive Bayesian classifiers (cf. Section 5.1.2) usually show a high precision (similar to the degree of precision achieved with LSI) and a much lower recall. In contrast to the investigations summarized in this chapter, Gee's approach is exclusively based on *textual* features of the email messages.

7.3 Open Issues and Own Contributions

One of the main motivations for this work was to investigate the influence of the choice of different feature sets on the performance of VSM- and LSI-based spam filtering. In particular, our objective was to extend Gee's approach (LSI on text-based features) to evaluate the classification performance achieved with LSI on a set of "meta features", such as those used in SpamAssassin, and to quantify the influence of feature set reductions on the classification results.

The investigation of LSI for spam filtering summarized here evaluates the relationship between two central aspects: (*i*) the truncation of the SVD in LSI and (*ii*) the resulting classification performance in this specific application context. It is shown that a surprisingly large amount of truncation is often possible without heavy loss in classification performance. This forms the basis for good and extremely fast approximate (pre-)classification strategies, which can be very useful in practice.

The low-rank approximation of the feature-document matrix within LSI is one possibility to reduce the potentially huge size of the problem. As an alternative, we investigate strategies for reducing the feature set prior to the classification/LSI process by using feature ranking methods introduced in Section 3.2. This pre-selection of high-ranked features can potentially reduce the computing time as the matrices in the time-consuming SVD computation become smaller. Classification performance of these two approaches to problem size reduction are investigated and compared in the following.

The approaches investigated in this chapter are shown to compare favorably to two competing approaches, in that they achieve better classification results than the SpamAssassin system, and they are better (in terms of classification accuracy) and more robust than the LSI-based approach using purely textual features proposed by Gee [Gee03].

7.4 Classification based on VSM and LSI

In text-mining applications, VSM and LSI represent a sample of documents in a *term-by-document* matrix. In the context of spam filtering, each document is an email message which is represented by a vector in the vector space model. In order to construct this VSM, each email has to be represented by a set of feature values determined by a feature extraction component. This leads to two central questions arising in this context: (*i*) Which features of email data are (most) relevant for the classification into spam and ham (textual features, header information, etc.) ? (*ii*) Based on a certain set of features, what is the *best* method for categorizing email messages into spam and ham ? In the remainder of this section, we will mainly focus on the former question. The evaluation of VSM and LSI in Section 7.5.2 contributes to answering the latter question.

7.4.1 Feature Sets

As already mentioned, we consider and compare two types of features which are very important and widely used in the context of spam filtering. On the one hand, we consider the features extracted by the state-of-the-art spam filtering system SpamAssassin. This feature set is denoted by "F_SA" in the following. On the other hand, we use a comparable number of purely text-based features. This feature set is denoted by "F_TB" in the following. First, these two types of feature sets and their extraction are discussed. Then, we discuss feature/attribute selection methods for investigating a controlled reduction of the number of features in each set.

SpamAssassin features. For feature set F_SA we use the SpamAssassin system (cf. Section 5.1.2) to extract their values from each message. SpamAssassin uses many tests and rules (795 in the version 3.1.1) to extract these features from each email message. Different parts of each email messages are tested, comprising the message body (56% of all tests) and the message header (36% of the tests). Moreover, there are URI checks (5% of the tests) as well as rawbody tests (MIME checks, 2% of the tests) and blacklist tests (1% of the tests). Tests acting on the message body comprise Bayesian classifiers, language specific tests, tests searching for obfuscated words, html tests and many others.[1] It should be mentioned that the SpamAssassin feature set is *"unbalanced"* in the sense that about 96% of all tests check for a feature which points to spam and only about 4% of the tests point to ham.

Each test leads to a certain feature value. This value is a positive real number if the test points to a spam message, and a negative real number if the test points to a ham message. The specific values (an thus weights for the features) have been derived by the SpamAssassin team using a neural network trained with error back propagation.[2] In the SpamAssassin system, the overall rating of a message is computed by summing up all values of the tests. If this sum exceeds a user-defined threshold (the standard threshold in version 3.1.x is set to 5) a message is classified as spam. Increasing the threshold will decrease the spam detection rate but will also reduce the false positive rate, whereas decreasing it will increase the spam detection rate but also the false positive rate.

Although the number of features extracted from each email message by the SpamAssassin system is very large, experimental analysis shows that only a relatively small subset of these features provides widely applicable useful information. More specifically, for our datasets only about *half* of all SpamAssassin tests triggered at least once, and only about 4% of the tests triggered for more than 10% of the messages (see Section 7.5.1). Besides the Bayesian classifier (which returns a value for every message) the SpamAssassin tests triggering most often for our datasets are the *"all_trusted"* test which checks whether a message has only passed through trusted SMTP relays (this test points to ham), as well as the *"razor2"* real-time blacklist test (this test points to spam). A more detailed list of the tests triggering most often for the datasets used in this work can be found in [GIL05]. As already mentioned, the SpamAssassin feature sets tends to be unbalanced. Another potential problem is that some of the SpamAssassin tests tend to trigger incorrectly. For our test data, eleven tests triggered wrongly for more than 2% of the messages, and seven of them triggered wrongly more often than correctly (see [GIL05]). Obviously,

[1]for a complete list see `http://spamassassin.apache.org/tests_3_1_x.html`

[2]`http://wiki.apache.org/spamassassin/HowScoresAreAssigned`

this tends to have a negative impact on the classification performance achieved. However, most of these problematic tests are assigned only a low default score and therefore their impact on the overall classification result is not too high in practice.

We selected those 377 and 299 SpamAssassin features, respectively, which triggered at least once for one of the datasets used (see Section 7.5.1) as a first starting point. We used the features in binary form and also the original values assigned by SpamAssassin. In most cases, using binary features turned out to yield slightly better classification results than using the original, weighted values. The experimental results (cf. Section 7.5) are based on binary feature values. The shortcomings of the SpamAssassin feature set mentioned above motivated our investigation of improved feature selection and extraction strategies summarized later in this chapter.

Text-based features. The classical alternative, which is used widely, not only in text mining, but also in the area of spam filtering, is a purely text-based feature set. Consequently, we consider a second feature set, which consists of specific words and tokens extracted from the email messages. *Document frequency thresholding* is used for reducing the potentially prohibitive dimensionality.

Document frequency thresholding. The processing of high dimensional data is a crucial issue when dealing with text documents or the content of textual email messages. Tens of thousands of features are not easy to handle, therefore a reduction of the feature space plays a significant role. The basic assumption is that very frequent terms (words, tokens) are less discriminative when trying to distinguish between classes (a term occurring in every single spam and ham message would not contribute to differentiate between them). Document frequency thresholding is an unsupervised feature selection method that achieves reductions in dimensionality by excluding terms having very high or very low document frequencies. Terms which occur in almost all documents in a collection do not provide any discriminating information. A good overview of various term selection methods explaining their advantages and disadvantages is given in [YP97]. Since document frequency thresholding is a feature selection method that is only applicable in text-mining applications, it is not explicitly listed in Chapter 3.

Document frequency thresholding is relatively inexpensive in terms of computational power. It proceeds as follows: First, the upper threshold is fixed to 0.5, hence all terms that occur in more than half of the documents are omitted. Then, the lower threshold is dynamically adapted in order to achieve a predefined desired number of features.

Furthermore, feature ranking techniques such as information gain (cf. Section 3.2.1) can be used to select the most important words from that "pre-selection" in order to further reduce computational cost. In constrast to document frequency thresholding, information gain is a supervised feature selection method, i.e., the feature selection process exploits the class label information of an object to measure if a feature is able to distinguish well between groups (classes) of objects (cf. Section 3.2).

7.4.2 Feature/Attribute Selection Methods Used

We used various feature/attribute selection methods to reduce the number of features within both feature sets introduced before. On that basis, we created different feature subsets comprising only top discriminating features for both feature sets. This feature reduction reduces the computational effort in the classification process for each message and therefore saves computing time.

We applied information gain attribute selection to the text-based as well as to the SpamAssassin features. The computation of the information gain for each feature can be performed similarly to the computation stated in Equations (3.1) and (3.2). Based on the resulting ranking of features, two subsets containing the top 50 and the top 10 features, respectively, were selected for both feature sets, F_SA and F_TB.

For the SpamAssassin feature set, we used two more feature selection strategies for comparison purposes – χ^2 attribute selection and an intuitive strategy: (*i*) The χ^2 attribute selection (cf. Section 3.2.1) evaluates the significance of an attribute by computing the value of the χ^2 statistic with respect to the class. (*ii*) As an intuitive strategy, we extracted two feature subsets containing only the top 50 and the top 10 triggering features, respectively (i. e., those SpamAssassin tests, which have triggered most often for all email messages within our training data). Since the computational runtimes for wrapper methods are usually much higher than for feature ranking methods (cf. Chapters 3 and 6), we did not use feature subsets selected by wrapper methods. Especially in the setting of time-critical on-line spam filtering this computationally complex feature selection step has counter-productive effects.

For the computation of the information gain and χ^2-attribute selection we used the publicly available machine learning software Weka (cf. [WF05]). Applying the χ^2-attribute selection to the text-based features, which have already been ranked by information gain did not change the order of the attributes/features. When comparing the three feature selection methods for the SpamAssassin feature set, eight SpamAssassin features occurred in all three top 10 subsets extracted. These comprise two Bayesian classification tests (*"bayes_00"* and *"bayes_99"*, indicating that the Bayesian spam probability is about 0% for *"bayes_00"* and about 99%-100% for *"bayes_99"*, respectively), black- and whitelist tests (*"razor2_cf_range_51_100"* and *"razor2_check"*), tests which check whether the message body contains an URL listed in a special URL blacklist (*"uribl_ws_surbl"* and *"uribl_ob_surbl"*), a test which checks whether all headers in the message were inserted by trusted SMTP relays (*"all_trusted"*), and a test which checks whether the message contains html tags (*"html_message"*).

7.4.3 Training and Classification

The application of VSM and LSI to spam filtering investigated here involves two phases: a training phase and the actual classification phase. The training phase comprises the indexing of two known datasets (one dataset consisting of spams and one dataset consisting of hams) and, in the case of LSI, the computation of the singular value decomposition of the feature- or term-document matrix. The classification phase comprises the query indexing and the retrieval of the closest message from the training sets. A newly arriving message can be classified by indexing it based on the feature set used and comparing the resulting query vector to the vectors in the training matrices. If the closest message is contained in the spam training set, then the query message is classified as spam, otherwise it is classified as ham. The distance measurement used is based on the angles between the query vector and the training vectors and is computed according to Equations (4.12) and (4.14) in Section 3.3.3.

7.5 Experimental Evaluation

The concepts and methods discussed before have been implemented and evaluated experimentally. The results achieved are summarized in this section.

LSI truncation. We denote the amount of approximation within the computation of the SVD (Equation (3.11)) by *LSI truncation*. The amount of truncation is given as a percentage value: If Σ_k contains only those k singular values of A which are greater than p percent of the maximum singular value of A, then the approximation is called "LSI p%" truncation. If, for example, the maximum singular value of A is 100, then for "LSI 2.5%" all k singular values which are greater than 2.5 are used in the approximation of A. Note that as the truncation parameter p increases, the rank of the approximation matrix decreases.

7.5.1 Datasets

We used two different datasets, one consisting of a part (25%) of the TREC 2005 spam/ham corpus. This dataset is denoted as "S1" in the following and is publicly available [CL05]. The other dataset consists of self collected email and is denoted as “S2”. The messages in S2 have been collected over a period of several months from various sources in order to achieve as much diversity as possible. Spam was collected from various spam traps, and ham came from volunteers (mainly private email). S1 contains 9 751 ham and 13 179 spam messages. S2 contains 5 502 ham and 5 502 spam messages.

As mentioned in Section 5.1.1, the problem of filtering spam is a binary classification problem in the sense that every incoming email has to be classified as “spam” or not spam (“ham”). The aggregated classification accuracy as well as the true positive (TP) and false negative (FP) rate are measured.

7.5.2 Experimental Setup

We performed two different cross validations (cf. Section 2.4.4) for our datasets, a threefold cross validation for the larger sample S1 and a tenfold cross validation for the smaller sample S2. The true/false positive rates and the true/false negative rates were measured and the aggregated classification results were computed from these results. The cross validations were performed for six different LSI truncations using the feature sets F_SA and F_TB as defined in Section 7.4.

7.5.3 Analysis of Data Matrices

The average ranks k of the truncated SVD matrices for both datasets and both feature sets are listed in Tables 7.1 and 7.2. The distribution of the singular values

Table 7.1: Rank k of the truncated SVD matrices for different cut-off values in the singular values for S1.

truncation:	0.0%	2.5%	5.0%	10.0%	25.0%	50.0%
			Features F_SA			
kHam:	377	24	12	5	1	1
kSpam:	377	94	47	23	6	1
			Features F_TB			
kHam:	377	83	37	14	4	3
kSpam:	377	86	50	19	7	2

in the vector space models for sample S1 is illustrated in Figures 7.1 and 7.2. The sharp decline in the magnitude of the singular values for both classes (ham and spam) is very interesting. The comparable figures for sample S2 are not depicted but show the same behavior for both feature sets, F_SA and F_TB.

Table 7.2: Rank k of the truncated SVD matrices for different cut-off values in the singular values for S2.

truncation:	0.0%	2.5%	5.0%	10.0%	25.0%	50.0%
			Features F_SA			
kHam:	299	31	15	7	3	1
kSpam:	299	79	42	17	5	2
			Features F_TB			
kHam:	299	53	26	14	4	2
kSpam:	299	55	27	11	4	2

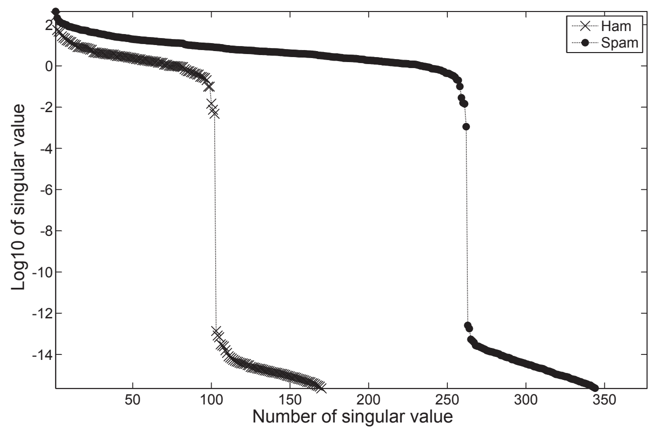

Figure 7.1: Singular values in vector space models for sample S1 using feature set F_SA.

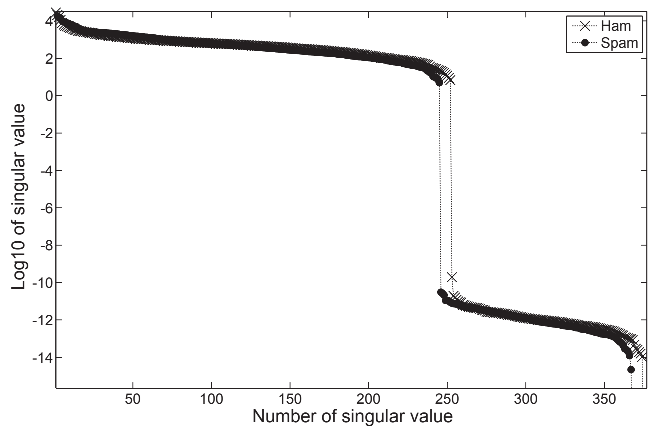

Figure 7.2: Singular values in vector space models for sample S1 using feature set F_TB.

Comparing Figures 7.1 and 7.2, it can be observed that the singular values for feature set F_TB tend to be significantly larger than the singular values for feature set F_SA. Until the sharp decline at about singular value number 250 (for both, ham and spam features), all singular values for the feature set F_TB are larger than 5, whereas the

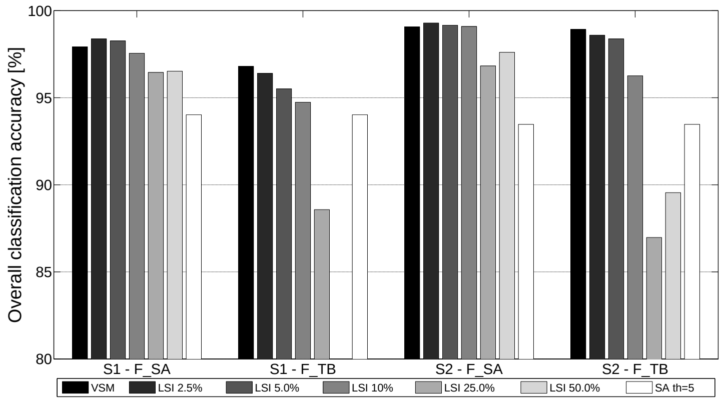

Figure 7.3: Aggregated classification results for S1 and S2.

majority of the singular values for feature set F_SA is much smaller than 5 (93% for the ham sample and 66% for the spam sample, respectively). Moreover, the largest singular values for the feature set F_TB are about 8 to 10 times larger than the largest singular values for the feature set F_SA. Looking at Figure 7.1, it is clearly visible that the singular values for the ham messages are significantly smaller than those for the spam messages.

This analysis of the data matrices forms the basis for understanding and explaining the observations summarized in the following sections: when using feature set F_SA, approximations of the feature-document matrix with a very low rank (even $k = 1$) still achieve a very good classification quality – partly even better than standard SpamAssassin (see the classification results in Figure 7.3 and the LSI truncation in Tables 7.1 and 7.2 for feature set F_SA and LSI 50%).

7.5.4 Aggregated Classification Results

Figure 7.3 depicts the overall (or aggregated) classification accuracy (cf. Sectio 2.4.5) for both datasets used. Six different LSI truncations are shown for the two feature sets F_SA and F_TB. As mentioned in Section 7.4.1, SpamAssassin assigns positive values to features pointing to spam and negative values to features pointing to ham. Classification is then performed based on a threshold value, the standard (and rather conservative) default value being 5. To compare our results to a standard approach the classification results of SpamAssassin using this default threshold are also shown in Figure 7.3 ("SA th=5"). The bar for the configuration [sample S1/LSI 50.0%/feature set F_TB] is not visible in the graph as in this case the overall classification accuracy is below 80%. It should be pointed out that in Figure 7.3 even for low LSI truncations most results are better than the overall classification results achieved by standard SpamAssassin.

When comparing the feature sets F_SA and F_TB, we observe that for all LSI truncations the classification results for F_SA are significantly better than the corresponding results for F_TB. For the LSI classification, the aggregated results for sample S2 are better than those achieved for S1. When using SpamAssassin with the default threshold 5 we observe the opposite – here, the results achieved for S1 slightly exceed the results achieved for S2. Using feature set F_SA, the aggregated results for LSI 2.5% and LSI 5.0% are even slightly better than the results for LSI 0.0% (basic VSM), where all features are used for classification. In contrast, for features F_TB almost every increase of the LSI truncation (decrease in the rank of the truncated SVD) causes a decrease in the classification accuracy (the only exception being [S2/LSI 50.0%]).

7.5.5 True/False Positive Rates

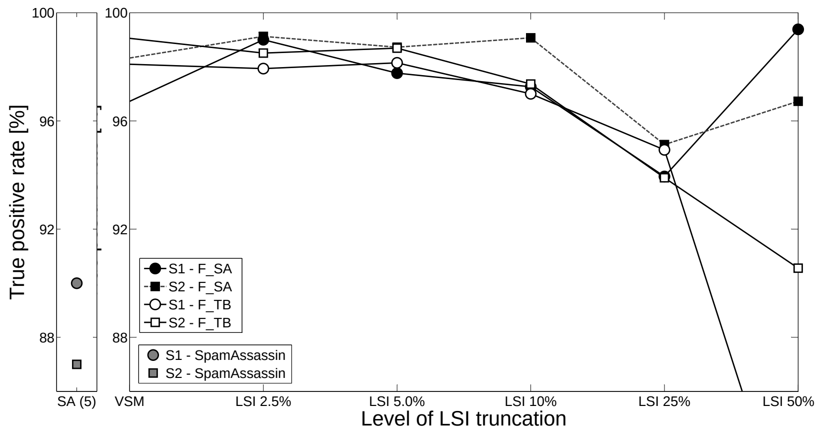

Figure 7.4: True positive rates.

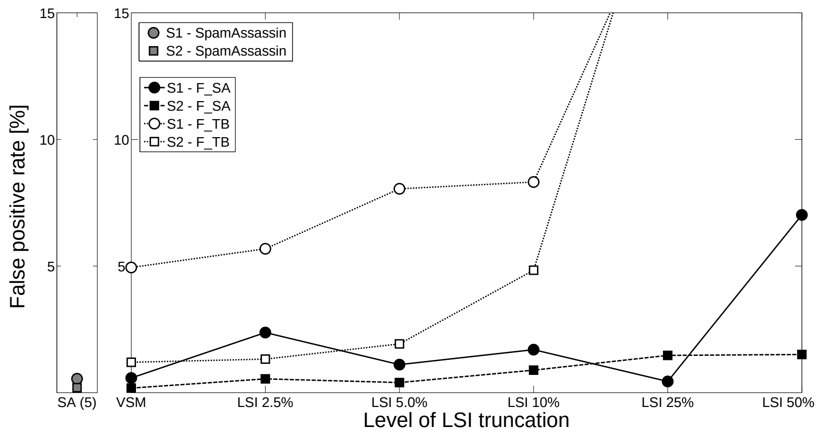

Figure 7.5: False positive rates.

In contrast to Figure 7.3, Figures 7.4 and 7.5 show true and false positive rates separately. The SpamAssassin threshold was again set to 5. Using a more aggressive threshold for SpamAssassin (< 5) would increase both the false positive rate *and* the true positive rate (e. g., for sample S1 a SpamAssassin threshold of 3 lead to a true/false positive rate of 93.89% and 1.94%, respectively).

Both figures show a similar behavior for the true/false positive rates for F_SA and F_TB and indicate a significantly higher false positive rate when using feature set F_TB. In particular, for sample S1 the false positive rate of F_TB is quite high (4.91% using VSM), although the aggregated classification accuracy reaches a respectable 96.80% (see Figure 7.3, F_TB using VSM). The false positive rate for sample S2 using feature set F_TB is slightly lower but still consistently higher than when feature set F_SA is used. The false positive rates tend to remain almost constant except for very high LSI truncations where they increase significantly.

7.5.6 Feature Reduction

The low-rank approximation of the feature-document matrix A within LSI is one way to reduce the problem size. Alternatively, one could try to reduce the number of features *before* the classification/LSI process using one of the feature selection methods mentioned in Section 7.4.1. Compared to the LSI truncation process on the full feature sets, this approach can significantly reduce the computing time as the time-consuming SVD computation within LSI is done for smaller matrices.

In order to compare these two basic approaches, we evaluated the classification results for various selected subsets of the feature sets F_SA and F_TB. These subsets were extracted using the feature/attribute selection methods mentioned in Section 7.4.1. When applying LSI on the reduced (information gain ranked, χ^2 ranked, and top triggering tests) feature sets, a combination of feature ranking (information gain, χ^2) followed by implicit dimensionality reduction (LSI) is performed to select the optimal feature subsets.

Feature reduction – aggregated results. Figure 7.6 shows the aggregated classification results using only top discriminating subsets of features from F_SA and F_TB, respectively, based on a ranking of features using information gain (see Section 7.4.1). The subsets selected contained the top 50 and the top 10 discriminating features, respectively. For classification, VSM and LSI 25% were used.

It can be observed that classification results based on a reduced number of features are slightly worse than the results achieved with the original feature sets (except for the top 50 F_TB features for sample S1, where the classification results are nearly equal). When applying LSI to reduced feature sets, only the results for

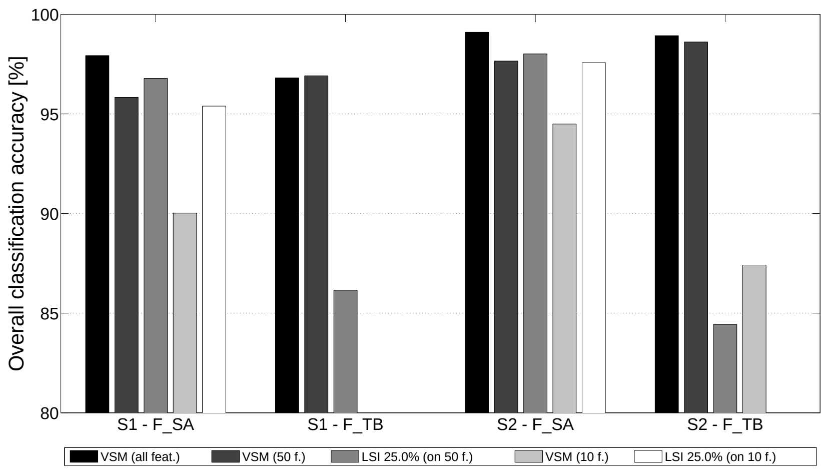

Figure 7.6: Aggregated results – VSM and LSI classification on reduced feature sets using information gain.

feature set F_SA remain acceptable. Thus, feature set F_SA turns out to be more robust in terms of feature reduction than feature set F_TB. It is also very interesting to notice that the LSI 25% classification based on ten selected features outperforms the VSM classification based on the same ten features for all sample/feature set combinations except S2–F_TB.

Feature reduction – true/false positive rates. Figures 7.7 and 7.8 show the true/false positive rates for samples S1 and S2 using different feature subsets from feature set F_SA. Three different attribute selection methods have been applied to extract these subsets (see Section 7.4.1).

While the classification results for the subsets selected by information gain and χ^2 attribute selection are almost equal (except for the true positive rates for sample S1 in Figure 7.7), the results for the subsets containing the top triggering tests (TT) differ slightly. The true/false positive rates for features F_TB are not depicted and are generally worse than the results for F_SA. Especially the false positive rates tend to increase strongly when using the reduced feature subsets.

All curves (except the TT true positive rate for sample S2 in Figure 7.7) show a similar behavior: When LSI 25% is applied (on the top 50 and top 10 feature subsets, respectively) the true positive *and* the false positive rates tend to increase in comparison to the basic VSM. Focussing on the classification results for VSM based on the top 50 and the top 10 features, respectively, it can be observed that for all subsets the false positive rates are nearly equal, whereas the true positive rates tend to decrease when the number of features is reduced. Interestingly, the false positive rates achieved with VSM based on 10 features are very low for both samples and all different feature subsets used (cf. Figure 7.8). Contrary to that,

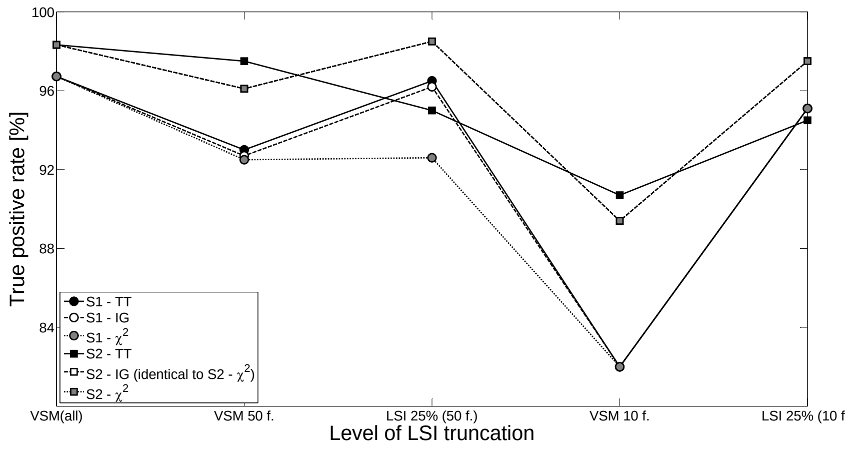

Figure 7.7: True positive rates for reduced feature sets using features F_SA.

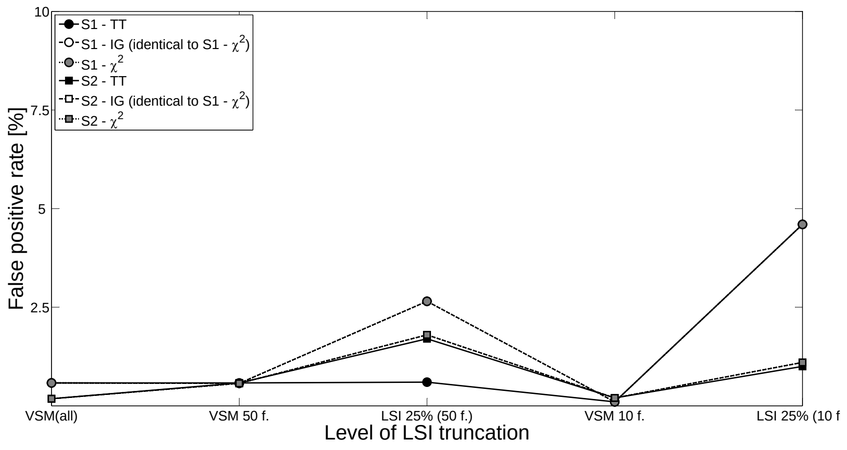

Figure 7.8: False positive rates for reduced feature sets using features F_SA.

the true positive rate achieved the same setting (VSM based on 10 features) tends to be worse than for other settings (e. g., LSI 25% (50 f.) and also LSI 25% (10 f.), cf. Figure 7.7). The results for LSI 25% based on 10 features are almost equal to the results of the same LSI truncation based on 50 features. Only for sample S1 the false positive rate for all three feature selection methods increases slightly when LSI 25% is based on only ten features.

7.6 Discussion

In this chapter, the application of latent semantic indexing to the task of spam filtering has been investigated. The underlying vector space models are based on two different feature sets: Purely text-based features resulting from word/token extraction (similar to the approach investigated by [Gee03]) were compared with a feature set based on the widespread SpamAssassin system for spam filtering. Moreover, several feature selection methods for extracting subsets of features containing the top discriminating features from each of the feature sets have been investigated as an alternative approach to reducing the size of the classification problem. Experimental evaluations on two large datasets showed several interesting results:

1. For both feature sets (F_SA and F_TB), VSM and LSI achieve significantly better classification results than the extremely widespread de-facto standard for spam filtering, SpamAssassin.

2. The classification results achieved with both VSM and LSI based on the feature set of SpamAssassin (F_SA) are consistently better than the results achieved with purely textual features (F_FB).

3. For both feature sets, VSM achieves better classification results than LSI at many truncation levels. However, when using LSI based on the SpamAssassin feature set, the classification results are surprisingly robust to LSI truncations with very low rank. Moreover, when applying VSM and LSI on reduced feature sets, LSI is very robust in terms of classification accuracy compared to VSM classification (see Figures 7.7 and 7.8). This indicates that LSI can provide ways for computing good (approximate) "pre-classifications" extremely fast, which is very useful in practice.

4. Distinguishing true and false positive rates, we observed that for all truncation levels of LSI the false positive rate, which is a very important performance metric in spam filtering, based on feature set F_SA is much lower than the one based on feature set F_TB. Standard SpamAssassin also achieves a good false positive rate, however, its true positive rate is rather poor compared to VSM and LSI (for the standard setting of SpamAssassin). Changing the SpamAssassin settings would result in a change in both true and false positive rate (cf. Section 7.5.5).

5. Among the approaches for reducing the problem size and thus the computational effort for the classification problem, LSI truncation based on the full feature sets turned out to be more stable than VSM classification using only subsets of the feature sets.

6. The classification performance of LSI based on SpamAssassin features is also quite robust if the number of features used is reduced significantly and still achieves remarkably good aggregated classification accuracy (Figures 7.7 and 7.8) which is not the case for LSI based on purely textual features (these results are not shown in the figures).

Overall, the experiments indicate that VSM and LSI are very well suited for spam filtering if the feature set is properly chosen. In particular, we showed that the SpamAssassin feature set achieves better results than the commonly used purely textual feature sets based on word- and token extraction. Both VSM and LSI based on properly chosen small subsets of the SpamAssassin feature set are very well suited as approximate but fast pre-classification strategies.

Chapter 8

New Initialization Strategies for Nonnegative Matrix Factorization

8.1 Overview of Chapter

In this chapter, we investigate new initialization strategies for nonnegative matrix factorization algorithms (NMF, cf. Section 3.3.4) based on a ranking of the original features. We apply the presented initialization strategies on data coming from the email classification context and compare our approach to standard random initialization and other initialization techniques for NMF described in the literature. Our approach shows faster reduction of the approximation error than random initialization and comparable or better results to existing but often more time-consuming approaches. Moreover, we consider the interpretability of the NMF factors in the email classification and drug discovery context and try to take advantage of information provided by the basis vectors in W (interpreted as basis emails or basis features, depending on the setup). Motivated by the approaches investigated in this chapter, we introduce new classification methods based on the NMF in Chapter 9.

8.2 Introduction and Related Work

NMF has been discussed in detail in Section 3.3.4. In short, NMF determines reduced rank *nonnegative* factors $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$ which approximate a given nonnegative data matrix $A \in \mathbb{R}^{m\times n}$, such that $A \approx WH$.

All algorithms for computing the NMF are iterative and require initialization of W and H (or at least of one of the factors). While the general goal – to establish initialization techniques and algorithms that lead to better overall error at convergence – is still an open issue, some initialization strategies can improve the NMF in terms of faster convergence and faster error reduction. Although the benefits of good NMF initialization techniques are well known in the literature, rather few algorithms for non-random initializations have been published so far.

Wild *et al.* [Wil02, WCD03, WCD04] were among the first to investigate the initialization problem of NMF. They used spherical k-means clustering based on the centroid decomposition (cf. [DM01]) to obtain a structured initialization for W. More precisely, they partition the columns of A into k clusters and select the centroid vectors for each cluster to initialize the corresponding columns in W (the number of clusters is identical to the NMF-parameter k). Their results show faster error reduction than random initialization, thus saving expensive NMF iterations. However, since this decomposition requires a clustering algorithm on the columns of A as a preprocessing step it is rather expensive (cf. [LMA06]).

[LMA06] have also provided some new initialization ideas and compared the aforementioned centroid clustering approach and random seeding to four new initialization techniques. While two algorithms (Random Acol and Random C) only slightly decrease the number of iterations and another algorithm (Co-occurrence) turns out to be very expensive computationally, the *SVD-Centroid* algorithm clearly reduces the approximation error and therefore the number of NMF iterations compared to random initialization. The algorithm is based on the SVD-centroid decomposition (cf. [CF01]), which is an approximation for the singular value decomposition. It can be formed by iteratively finding the centroid factors and their corresponding factor loadings. Each step proceeds by finding the most significant centroid factor and using it to perform a rank reduction on the original matrix (cf. [BC04]). More precisely, the algorithm initializes W based on a centroid decomposition of the low dimensional SVD factor $V_{n\times k}$, which is much faster than a centroid-decomposition on $A_{m\times n}$ since V is much smaller than A (cf. [LMA06]). Nevertheless, the SVD factor V must be available for this algorithm, and the computation of V can obviously be time-consuming.

[BG08] initialized W and H using a technique called *nonnegative double singular value decomposition* (nndsvd) which is based on two SVD processes utilizing an algebraic property of unit rank matrices – nndsvd is based on the Perron-Frobenius Theorem [CHN04] which guarantees nonnegativity for the first left and right singular vectors of a nonnegative matrix. The first SVD process approximates the data matrix A by computing the first k singular values and corresponding singular vectors. The second SVD process iteratively approximates positive sections of the resulting partial SVD factors. The authors performed various numerical experiments and showed that nndsvd initialization is better than random initialization in term of faster convergence and error reduction in all test cases, and generally appears to be better than the centroid initialization in [Wil02] for most algorithms. However, results in [BG08] show that the gradient descent algorithm achieves a faster error reduction when using the centroid initialization [Wil02] instead of nndsvd.

8.3 Open Issues and Own Contributions

Despite the fact that some initialization strategies have been published, in practice randomized seeding of W and H is still the standard approach for many NMF algorithms. Existing approaches, such as the ones mentioned in Section 8.2 can be rather time consuming. Obviously, the trade-off between computational cost in the initialization step and the computational cost of the actual NMF algorithm needs to be balanced carefully. In some situations (for example, for most initialization strategies introduced in Section 8.2), an expensive preprocessing step may overwhelm the cost savings in the subsequent NMF update steps.

In this chapter, we introduce a simple and fast initialization step based on feature subset selection (FS, cf. Section 3.2) and show comparisons with random initialization and the nndsvd approach mentioned before. More precisely, we use information-theory based feature ranking based on information gain and gain ratio. We use implementations of the multiplicative update (MU) algorithm and the alternating least squares (ALS) algorithm (these algorithms do not depend on a step size parameter, as it is the case for gradient descent) from the statistics toolbox v7.1 in Matlab (included since R2009a release). The MU and the ALS algorithm are the two available algorithms included in this toolbox. A description of these NMF algorithms can be found in Section 3.3.4. We used the same termination criteria for both algorithms as implemented in Matlab.

8.4 Datasets

Two different datasets – coming from email classification and drug discovery problems – were used for evaluation in this chapter. The drug discovery dataset consists of 249 structurally diverse chemical compounds and is identical to the drug discovery dataset used for evaluation in Chapter 6.

The email dataset consist of 15 000 email messages, divided into three groups – ham, spam and phishing (note that this is a different dataset, than the one used in Chapters 6 and 7). The email messages were taken partly from the *Phishery* [Phi09] and partly from the 2007 TREC corpus [NIS07]. The email messages are described by 133 features (purely text-based features, online features and features extracted by rule-based filters such as SpamAssassin (cf. Section 5.1.2)). Since the features in this dataset are not solely computed by the SpamAssassin system and thus differ in their range, we applied a different preprocessing step than in Chapter 7. Here we scaled all feature values to [0,1] to ensure that they have the same range.

The structure of phishing messages tends to differ significantly from the structure of spam messages, but it may be quite close to the structure of regular ham messages (because for a phishing message it is particularly important to look like a regular message from a trustworthy source). feature set has been given in [GP09].

The email corpus was split up into two sets (for training and for testing), the training set consisting of the oldest 4 000 email messages of each class (12 000 messages in total), and the test set consisting of the newest 1 000 email messages of each class (3 000 messages in total). This chronological ordering of historical data allows for simulating the changes and adaptations in spam and phishing messages which occur in practice. Both email sets are ordered by the classes – the first group in each set consists of ham messages, followed by spam and phishing messages. Due to the nature of the features the data matrices are rather sparse. The bigger (training) set has 84.7% zero entries, and the smaller (test) set has 85.5% zero entries.

8.5 Interpretation of Factors

A key characteristic of NMF is the representation of basis vectors in the factor W and the representation of basis coefficients in the factor H. With these coefficients the columns of A can be represented in the basis given by the columns of W. In the context of email classification, W may contain *basis features* or *basis emails*, depending on the structure of the original data matrix A. If NMF is applied to an *email* $\times$ *feature* matrix (i. e., every row in A corresponds to an email message), then W contains k basis *features*. If NMF is applied on the transposed matrix (*feature* $\times$ *email* matrix, i. e., every column in A corresponds to an email message), then W contains k basis *email messages*.

Similarly, in the context of classifying drug molecules or compounds as substrates or non-substrates, W may contain *basis descriptors* or *basis compounds/molecules*, depending on the structure of the original data matrix A. If NMF is applied to a *compound* $\times$ *descriptor* matrix (i. e., every row in A corresponds to a compound), then W contains k basis *descriptors*. If NMF is applied on the transposed matrix (*compound* $\times$ *descriptor* matrix, i. e., every column in A corresponds to an compound), then W contains k basis *compound*.

8.5.1 Interpretation of Email Data

Basis features. Figure 8.1 shows three basis features $\in \mathbb{R}^{12\,000}$ (for k=3) for the training set of the email dataset when NMF is applied to an *email* $\times$ *feature* matrix. The three different groups of objects – ham (first 4 000 messages), spam (middle 4 000 messages) and phishing (last 4 000 messages) – are easy to identify. The group of phishing email tends to yield high values for basis feature 1, while basis feature 2 shows the highest values for the spam messages. The values of basis feature 3 are generally smaller than those of basis features 1 and 2, and this basis feature is clearly dominated by the ham messages.

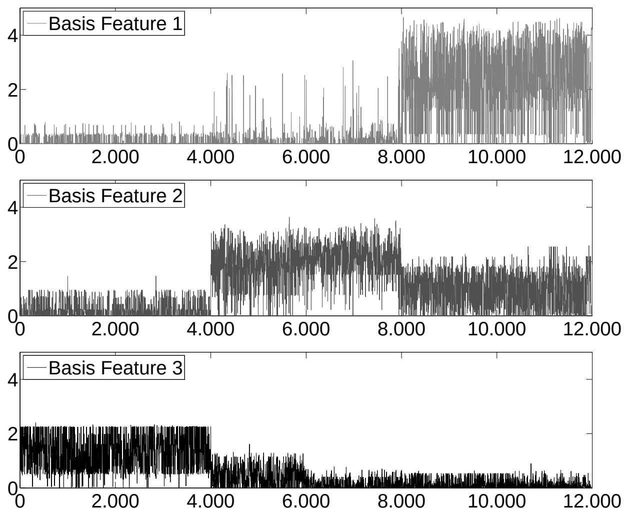

Figure 8.1: Basis features for $k = 3$.

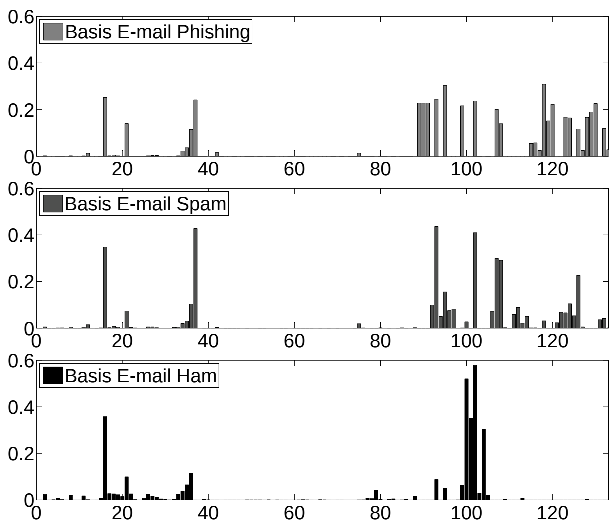

Figure 8.2: Basis email messages for $k = 3$.

Basis email messages. The three basis email messages $\in \mathbb{R}^{133}$ (again for k=3) resulting from NMF on the transposed (*feature* $\times$ *email*) matrix are plotted in Figure 8.2. The figure shows two features (#16 and #102) that have a relatively high value in all basis emails, indicating that these features do not distinguish well between the three classes of email. These tests are called *"HtmlMailTest"* and *"to_malformed"*, respectively, and check if the email is an HTML email (#16) and if the TO field has a malformed address (#102). Other features better distinguish between classes. For example, features 89-91 and 128-130 have a high value in basis email 1, and are (close to) zero in the other two basis emails. Investigation of the original data shows that these features tend to have high values for phishing email – indicating that the first basis email represents a phishing message. These tests are all integrated into SpamAssassin [Spa09] and are called *"dc_gif_uno_largo"*, *"dc_image_spam_html"*, *"dc_image_spam_test"*, *"base64_lenght_79_inf"*, *"from_local_digits"*, *"from_local_hex"*, *"helo_lh_ld"*, and *"spoof_com2oth"*.

Using the same procedure, the third basis email can be identified to represent ham messages (indicated by features 100 and 101). These two features are again SpamAssassin tests and are called *"rcvd_in_dnswl_med"* and *"spf_helo_pass"*. Finally, basis email 2 represents spam.

8.5.2 Interpretation of Drug Discovery Data

In this section we make some comments on the interpretability of NMF factors W and H in the drug discovery context. Contrary to the ternary email classification problem mentioned before, the classification of molecules into substrates/non-substrates is a binary classification problem. Thus, we set the number of basis descriptors/compounds under investigation equal to the number of classes (k=2).

Basis compounds. Figure 8.3 shows two basis compounds (abbreviated as *BC* in the following) $\in \mathbb{R}^{366}$ (again for k=2) when NMF is applied to a (*descriptor* $\times$ *compound*) matrix. The difference of descriptor values between first and second BC is also shown in Figure 8.3 (subfigure "Diff (BC1-BC2)"). The subfigure is scaled such that the limits on the y-axis (+/-100%) indicate the greatest difference (absolute value) between BC1 and BC2 (in this case descriptor #184). More precisely, the values of "Diff (BC1-BC2)" are computed as $(\mathrm{BC1}_i - \mathrm{BC2}_i) * 100/\mathit{maxdiff}$, where *maxdiff* is the maximum absolute value over all $\mathrm{BC1}_i - \mathrm{BC2}_i$. Moreover, the different groups of chemical descriptors used in this dataset are shown.

It can be seen that several groups of descriptors show very different characteristics. While some groups of descriptors tend to have higher values for BC1 (2DACorr, most geometric descriptors,...), other groups tend to have higher values for BC2

(ACD). This indicates, that several groups of descriptors tend to represent a specific group of compounds. In the ideal setting one basis compound represents substrates, while the other basis compound represents non-substrates. However, unfortunately this cannot be guaranteed. Although several groups of descriptors tend to have different values within different BCs, only few descriptors show a significant difference between them. As can be seen in Figure 8.3, only 9 descriptors exceed the +/-50% marks (which indicates that the difference of descriptor x between both BCs is larger than 50% of the maximal difference of all descriptors.)

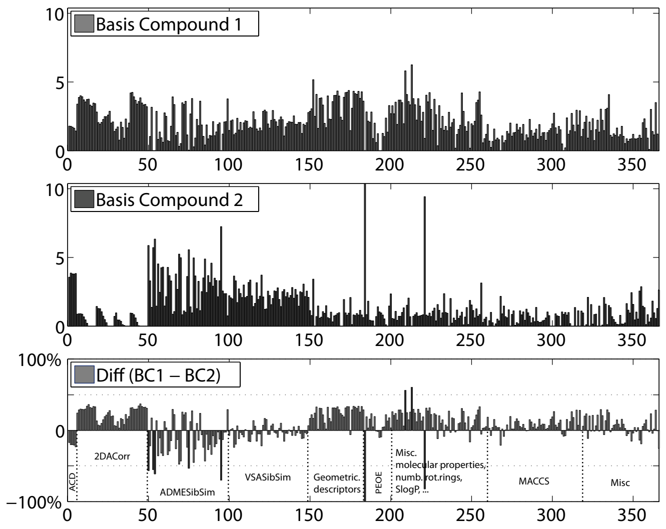

Figure 8.3: Basis compounds for $k = 2$.

To increase the number of descriptors having significant differences between BCs we apply sparseness constraints for the NMF as introduced in a study by Hoyer [Hoy04]. This study defines a sparseness measure for the NMF factors and describes a projection operator capable of enforcing any given sparseness for both, W and H. Figure 8.4 shows again two basis compounds $\in \mathbb{R}^{366}$ (for k=2), this time applying sparseness constraints. For both NMF factors W and H the sparseness level was set to 0.4 (i. e., 40% of all entries in W and H, respectively, are zero). As can be seen, the BCs are very different compared to Figure 8.3. The difference of the descriptor values "Diff (BC1-BC2)") between the two BCs changed significantly. The group of ACD descriptors and the group of geometrical descriptors now have larger values in BC2, while the group of VSASibSim descriptors shows very high values in BC1 and almost only zero entries in BC2. Moreover, there is a greater difference in the descriptor values between BC1 and BC2, compared to a factorization without sparseness constraints. Overall, 97 descriptors exceed the +/-50% marks. Obviously, when applying sparseness contraints, more descriptors show a significant difference between the two basis compounds.

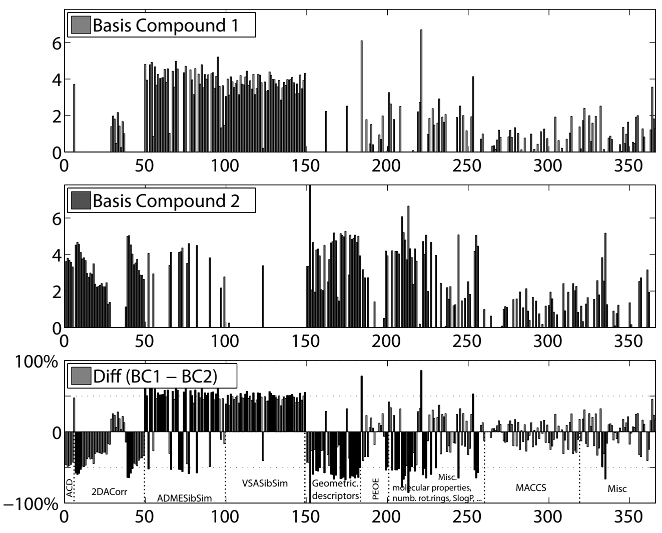

Figure 8.4: Basis compounds for $k = 2$ using sparseness constraints.

This rich structure observed in the basis vectors should be exploited in the context of classification methods (cf. Chapter 9). However, the structure of the basis vectors heavily depends on the concrete feature set used. In the following, we discuss the application of feature selection techniques in the context of NMF initialization.

8.6 Initializing NMF

As already mentioned in Section 8.2, the initialization of the NMF factors has a big influence on the speed of convergence and on the error reduction of NMF algorithms. Existing approaches such as initialization based on spherical k-means clustering [Wil02] or nonnegative double singular value decomposition (nndsvd) [BG08] can be rather costly in terms of runtime. As an alternative, we introduce a simple and fast initialization step based on feature subset selection and show comparisons with random initialization and the nndsvd approach mentioned before, which has shown to achieve the best results amongst the existing initialization strategies (cf. Section 8.2). For experimental evaluation in this section we used the email dataset described in Section 8.4.

8.6.1 Feature Subset Selection

As summarized in Section 3.2.1, the main idea of univariate filter methods is to rank features according to how well they differentiate between object classes. Redundant or irrelevant features can be removed from the dataset as they can lead to a reduction of the classification accuracy or clustering quality and to an unnecessary increase of computational cost. The output of this process is a ranking of features based on the applied filter algorithm. The two filter methods used in this chapter are based on *information gain* (IG) and *gain ratio* (GR), both reviewed in Section 3.2.1.

8.6.2 FS-Initialization

After determining the feature ranking based on IG and GR, we use the k first ranked features to initialize W (denoted as *FS-initialization*). Since feature selection aims at reducing the feature space, our initialization is applied in the setup where W contains basis features (i. e., every row in A corresponds to an email message, cf. Section 8.5). FS methods are usually computationally rather inexpensive (cf. Chapter 6 or [JGD08] for a comparison of IG and PCA runtimes) and can thus be used as a computationally relatively cheap but effective initialization step.

8.6.3 Results

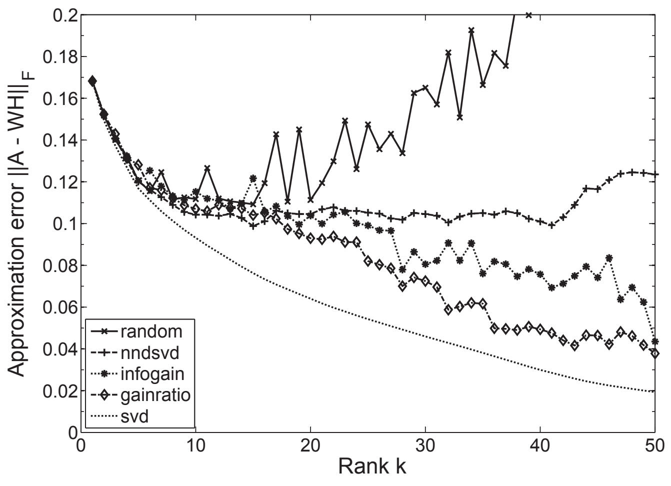

Figure 8.5: Approximation error for different initialization strategies using the ALS algorithm (*maxiter*=5).

Figures 8.5 and 8.6 show the NMF approximation error for our new initialization strategy for both information gain (infogain) and gain ratio feature ranking as well as for nndsvd and random initialization when using the ALS algorithm. As a baseline, the figures also show the approximation error based on an SVD of A, which gives the best possible rank k approximation of A (cf. Section 3.3.3). For rank k=1, all NMF variants achieve the same approximation error as the SVD, but for higher values of k the SVD has a smaller approximation error than the NMF variants (as expected, since SVD gives the best rank k approximation in terms of

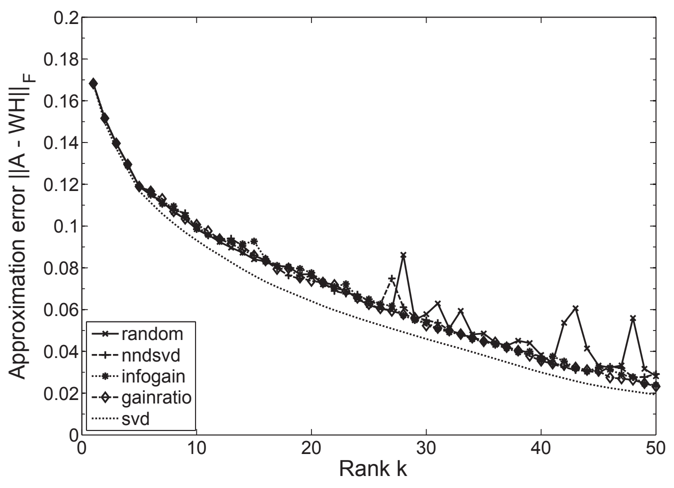

Figure 8.6: Approximation error for different initialization strategies using the ALS algorithm (*maxiter*=30).

approximation error). Note that when the maximum number of iterations inside a single NMF factorization (*maxiter*) is high (*maxiter*=30 in Figure 8.6), the approximation errors are very similar for all initialization strategies used and are very close to the best approximation computed with SVD. However, with a small number of iterations (*maxiter*=5 in Figure 8.5), it is clearly visible that random seeding cannot compete with initialization based on nndsvd or feature selection. Moreover, for this small value of *maxiter*, the FS-initializations (both information gain and gain ratio ranking) show better error reduction than nndsvd with increasing rank k. With increasing values of *maxiter* the gap between the different initialization strategies decreases until the error curves become almost identical when *maxiter* is about 30 (see Figure 8.6).

8.6.4 Runtime

In this Section we analyze runtimes for computing rank k NMF for different initialization strategies and different values of *maxiter* using the ALS algorithm. We compare the runtimes needed to achieve a specific approximation error based on a given initialization strategy. The computational cost for computing the initialization strategies are not compared, since different initializations of the factors were performed with different software.

The Matlab implementation of the ALS algorithm is not the best implementation in terms of runtime. This is due to the fact that the specific algorithm used for solving the simultaneous linear equations $XA = B$ depends upon the structure of the coefficient matrix A. In this work, we computed the ALS update steps (see

Table 8.1: Runtime comparison (n.a. = not available.

$\|\|A-WH\|\|_F$	***maxiter* 5**	***maxiter* 10**	***maxiter* 15**	***maxiter* 20**	***maxiter* 25**	***maxiter* 30**
			Gain ratio initialization			
0.10	0.6s (k=17)	1.0s (k=11)	1.5s (k=11)	2.0s (k=11)	2.2s (k=10)	2.7s (k=10)
0.08	0.9s (k=27)	1.5s (k=22)	2.2s (k=21)	2.9s (k=19)	3.1s (k=19)	3.3s (k=19)
0.06	1.1s (k=32)	2.0s (k=30)	2.8s (k=28)	3.7s (k=28)	4.6s (k=27)	5.4s (k=26)
0.04	**1.5s** (k=49)	2.4s (k=40)	3.9s (k=40)	5.0s (k=40)	6.3s (k=38)	7.2s (k=38)
			Information gain initialization			
0.10	0.6s (k=18)	1.0s (k=12)	1.6s (k=12)	1.8s (k=10)	2.2s (k=10)	2.7s (k=10)
0.08	1.0s (k=28)	1.5s (k=22)	2.3s (k=22)	2.9s (k=19)	3.1s (k=19)	3.3s (k=19)
0.06	1.5s (k=48)	2.0s (k=30)	3.0s (k=30)	3.7s (k=28)	4.6s (k=28)	5.4s (k=26)
0.04	**1.6s** (k=50)	2.5s (k=42)	4.1s (k=42)	5.1s (k=41)	6.3s (k=38)	7.2s (k=38)
			nndsvd initialization			
0.10	0.6s(k=15)	1.0s (k=12)	1.6s (k=12)	1.8s (k=10)	2.2s (k=10)	2.7s (k=10)
0.08	**n.a.**	1.7s (k=25)	2.6s (k=25)	2.6s (k=18)	3.1s (k=19)	3.2s (k=18)
0.06	**n.a.**	2.1s (k=32)	3.1s (k=32)	3.9s (k=29)	4.6s (k=28)	5.7s (k=30)
0.04	**n.a.**	**n.a.**	**n.a.**	**5.1s** (k=41)	6.3s (k=38)	7.2s (k=38)
			Random initialization			
0.10	**n.a.**	0.9s (k=10)	1.4s (k=10)	1.8s (k=10)	2.2s (k=10)	2.7s (k=10)
0.08	**n.a.**	1.5s (k=22)	2.3s (k=22)	2.5s (k=17)	3.1s (k=19)	3.3s (k=19)
0.06	**n.a.**	**n.a.**	**n.a.**	4.1s (k=31)	4.5s (k=26)	5.4s (k=26)
0.04	**n.a.**	**n.a.**	**n.a.**	**5.4s** (k=45)	6.7s (k=42)	7.3s (k=39)

Algorithm 3) using an economy-size QR-factorization, i. e., only the first n columns of the QR-factorization factors Q and R are computed (here n is the smaller dimension of the original data matrix A). This adaptation of the Matlab ALS algorithm has been implemented in Matlab and shows to save computation time (about 3.7 times faster than the original ALS algorithm implemented in Matlab), while achieving the same results.

Generally, the algorithms terminated after the number of iterations reached the pre-defined threshold *maxiter*, i. e., the approximation error and the relative change from the previous iteration were not integrated in the stopping criterion. Consequently, the runtimes do not depend on the initialization strategy used (neglecting the marginal runtime savings due to sparse initializations). In this setup, a linear relationship between runtime and the rank of k can be observed. Consequently, reducing the number of iterations (lower values of *maxiter*) brings important reductions in runtimes. This underlines the benefits of our new initialization techniques: As Figure 8.5 has shown, our FS-initialization reduces the number of iterations required for achieving a certain approximation error compared to existing approaches.

Table 8.1 compares runtimes needed to achieve different approximation error thresholds with different values of *maxiter* for different initialization strategies. It illustrates that a certain approximation error $||A - WH||_F$ can be achieved much faster with small *maxiter* and high rank k than with high *maxiter* and small rank k. As can be seen in Table 8.1, an approximation error of 0.04 or smaller can be computed in 1.5 and 1.6 seconds, respectively, when using gain ratio and information gain initialization (here, only 5 iterations (*maxiter*) are needed to achieve an approximation error of 0.04.). To achieve the same approximation error with nndsvd or random initialization, more than five seconds are needed (here, 20 iterations are needed to achieve the same approximation error). These results are highlighted in bold font in Table 8.1. The abbreviation `n.a.` (not available) indicates that a specific approximation error could not be achieved with the given value of *maxiter*.

8.7 Discussion

In this chapter, we have investigated a fast initialization technique for nonnegative matrix factorization (NMF). Our approach is based on feature subset selection (FS-initialization) which significantly reduces the approximation error of the NMF compared to randomized seeding of the NMF factors W and H. More specifically, we applied two feature ranking methods – information gain and gain ratio – and used the k top ranked features to initialize the NMF factor W (in this setting W contains basis features). Comparison of our approach with existing initialization strategies such as nndsvd [BG08] shows basically the same accuracy when many NMF iterations are performed, but much better accuracy when the NMF algorithm is restricted to a small number of iterations (based on the value of *maxiter*). The gap in approximation accuracy between the different initialization strategies increases with increasing rank k, especially the results achieved with random initialization are relatively poor for higher rank k (cf. Figure 8.5). As a consequence of the faster reduction in approximation error, the runtime needed to achieve a certain approximation accuracy can be reduced significantly with our initialization approach.

The interpretability of the NMF factors W and H in the email classification context was another important issue of this chapter. We tried to take advantage of the information provided by the basis vectors in W. Depending on the structure of the original data matrix A, the basis vectors (columns) in W represent either basis features or basis emails. Investigations of the basis features show that – when setting k equal to the number of distinct classes of email (3) – each basis email represents exactly one group of email messages (ham, spam, phishing). Features with high values in one basis email and small or zero values in the other two basis emails can obviously distinguish very well between classes and are thus more applicable for classifying unlabeled email than features having high values in all basis emails (cf. Section 8.5). Consequently, this approach can be used as a NMF-based feature selection method, which selects important features for discriminating between classes of objects in an unsupervised manner.

Chapter 9

Utilizing Nonnegative Matrix Factorization for Classification Problems

9.1 Overview of Chapter

In this chapter, we analyze the classification accuracy and the computational cost of different classification methods based on nonnegative matrix factorization (NMF). Based on the four different initialization strategies discussed in Chapter 8, we investigate different classification algorithms which utilize NMF for developing a classification model.

In the first part of this chapter, we analyze the classification accuracy achieved with the basis features in W when initialized with the techniques explained in Chapter 8. Since basis vectors (columns of W) of different NMF factorizations are difficult to compare with each other, in this setting the NMF is computed on the complete dataset (training and test data). Thus, this technique can only be applied on data that is already available before the classification model is built. However, instances of the test dataset must not be labeled prior to the NMF.

In the second part of this chapter, we introduce a new classifier based on NMF which can be applied dynamically to new data. Here the factorization of the data (NMF) and the classification process are separated from each other (i. e., the NMF is performed on labeled training data – the unlabeled test data must not be available at the time of performing the NMF). We also present a combination of NMF with latent semantic indexing (LSI) and compare it to standard LSI based on singular value decomposition (SVD).

9.2 Introduction and Related Work

The utilization of low-rank approximations in the context of email classification has been analyzed several studies. Gee [Gee03] has investigated the classification performance of LSI applied to the task of binary spam filtering. The results show that LSI enjoys a very high degree of recall and also a high degree of precision. In [GJN08], LSI was applied successfully on both purely textual features and on features extracted by rule-based filtering systems. Especially the features from rule-based filters allowed for a strong reduction of the dimensionality without loosing significant accuracy in the classification process. The results of this study are also mentioned in this thesis in Chapter 7.

NMF has been used for in combination with supervised classification methods for several application areas, for example, the classification of images (cf. [GSV02, GVS03]), musical instruments (cf. [BKK06]), or tumors (cf. [ZZL08]). Berry and Browne [BB05a] have demonstrated how a specific NMF algorithm can be used for extracting and tracking topics of discussion from corporate email. In their work they used the gradient descent algorithm (cf. Section 3.3.4) extended with least squares constraints. This extension of the GD algorithm was first presented in Shahnaz *et al.* [SBP06]. NMF was used to cluster topics of email communication and the authors showed how a parts-based NMF representation of corporate email can facilitate the observations of email discussions without requiring human intervention or the reading of individual messages.

9.3 Open Issues and Own Contributions

As mentioned before, the potential of NMF to cluster topics of textual documents such as the content of email messages has been analyzed (cf. [BB05a]), and several studies mentioned in Section 9.2 have investigated the application of classification methods based on low-rank approximations achieved with NMF. Moreover, it has been shown that low-rank approximations based on SVD are able to achieve good classification results when applied to the task of email classification (cf. [Gee03, GJN08, JGD08]). However, no studies can be found that analyze the classification accuracy achieved with a classification model performing on the NMF factors W and H. In this setting, NMF can be used as dimensionality reduction technique without the need to recompute and store the complete approximation matrix (i. e., the product WH).

In this chapter, we investigate the classification accuracy achieved with the basis vectors (features) in W when initialized with the techniques explained in Chapter 8.

We use a support vector machine classifier to investigate the influence of different initialization strategies on the classification accuracy. We analyze the classification performance achieved with different values of *maxiter* (maximum number of iterations within a single NMF factorization) and show that the value of *maxiter* has a strong influence on the computational cost of the factorization, and also on the classification accuracy, especially when using random initialization of the factors W and H.

Moreover, in Section 9.5 we introduce a new classification approach based on NMF which can be applied to a setting where new data becomes available dynamically. In this setting, the NMF is computed on the training data and unlabeled data can be classified without recomputing the factorization. Obviously, this case is more suitable for practical systems since the computation of the NMF can be rather costly in terms of runtime and computational cost. We also present a combination of NMF with latent semantic indexing (LSI, cf. Section 4.8). Our LSI-based classification approach is not based on singular value decomposition (SVD, cf. Section 3.3.3) as it is the case for standard LSI, but on the NMF factors W and H. Besides a comparison of the classification accuracy achieved with different generalizations of LSI we analyze the classification runtimes for all LSI variants. The runtimes comprise two steps – the computation of the low-rank approximation (SVD, NMF) and the classification process.

9.4 Classification using Basis Features

Figures 9.1 and 9.2 show the overall classification accuracy for a ternary classification problem (same datasets discussed in Section 8.4) using different values of *maxiter* for all four initialization strategies mentioned in Section 8.6. As classification algorithm we used a support vector machine (SVM, cf. Section 4.5) with a radial basis kernel provided by the MATLAB LIBSVM (version 2.88) interface (cf. [CL01]). For the results shown in this section, we performed 5-fold cross validation on the larger email corpus (consisting of 12 000 email messages, cf. Section 8.4).

The results based on the four NMF initialization techniques (infogain, gainratio, nndsvd and random) were achieved by applying an SVM on the rows of W. If the original data matrix $A \in \mathbb{R}^{m\times n}$ is an email $\times$ feature matrix, then the NMF factor W is a $m \times k$ matrix, where every email message is described by k basis *features*, i. e., every column of W corresponds to an basis feature (cf. Section 8.5). We thus built an SVM model on the rows of W instead on the rows of A. The results are shown for the multiplicative update algorithm (MU, cf. Section 3.3.4), but the results achieved with other NMF algorithms are similar. For comparison with the original features,

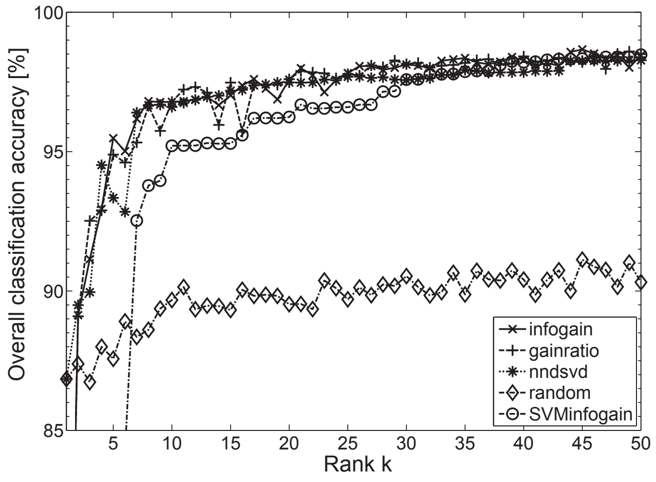

Figure 9.1: SVM (RBF-kernel) classification accuracy for different initialization methods using the MU algorithm (*maxiter*=5).

we also applied a standard SVM classification on the email messages characterized by k best ranked information gain features (denoted as "SVMinfogain" in Figures 9.1 and 9.2). The graph for SVMinfogain is identical in both figures since the *maxiter* factor in the NMF algorithm has no influence on the result.

Classification results. For lower ranks ($k < 30$), the SVMinfogain results are markedly below the results achieved with non-randomly initialized NMF (infogain, gainratio and nndsvd). This is not very surprising, since W contains compressed information about all features (even for small ranks of k). Random NMF initialization of W (random) achieves even lower classification accuracy for *maxiter*=5 (see Figure 9.1). The classification result with random initialization remains unsatisfactory even for large values of k. With larger *maxiter* (cf. Figure 9.2), the classification accuracy for randomly seeded W increases and achieves results comparable to infogain, gainratio and nndsvd. Comparing the results of the FS-initialization and nndsvd initialization it can be seen that there is no big gap in the classification accuracy. It is interesting to notice the decline in the classification accuracy of nndsvd for k=6 (in both figures). Surprisingly, the classification results for *maxiter*=5 are only slightly worse than for *maxiter*=30 – which is in contrast to the approximation error results shown in Section 8.6. Consequently, fast (and accurate) classification is possible even for small *maxiter* and small k (for example, the average classification accuracy over infogain, gainratio and nndsvd is 96.75% for k=10 and *maxiter*=5, compared to 98.34% for k=50, *maxiter*=30).

Classification results for drug discovery problems. Experimental evaluation of the classification approach when using basis features/descriptors for drug

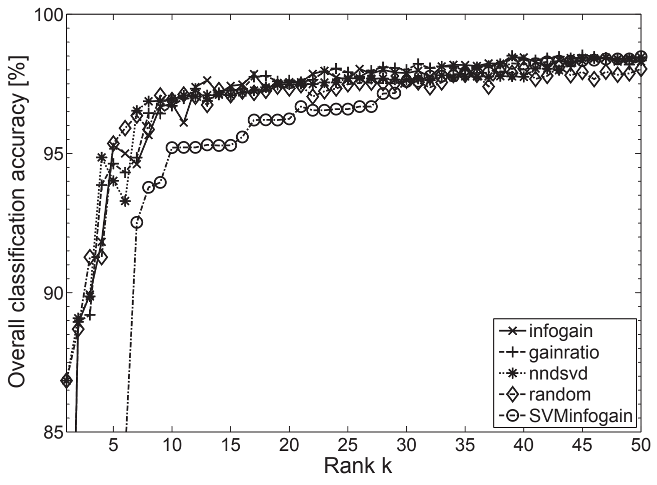

Figure 9.2: SVM (RBF-kernel) classification accuracy for different initialization methods using the MU algorithm (*maxiter*=30).

discovery problems also revealed interesting results. Classification results achieved with the drug discovery dataset mentioned in Section 8.5.2 using three different classification algorithms (SVM, J.48 and kNN, cf. Chapter 4) implemented in Weka (cf. [WF05]) showed that for some learning algorithms the classification accuracy could be improved significantly when applying the classification approach introduced in this section. Table 9.1 summarizes the results and can be interpreted as follows: The baseline is set by a classification based on the original feature set (original descriptors). The results for different values of k indicate if the classification results could be improved when classifying using the NMF factor W or not. The results were achieved with IG initialization of W. Surprisingly, the overall classification accuracy achieved with specific learning algorithm can be improved significantly when using only 2 basis features (i. e., basis descriptors), for example, when using SVM or the J.48 decision tree. However, for other learning algorithms such as kNN, the classification accuracy decreased.

Table 9.1: Classification results using basis features for drug discovery problems.

	SVM	J.48	kNN
Original features	*64.3%*	*60.9%*	*66.3%*
k=2	+3.5%	+7.4%	-6.0%
k=4	+3.0%	+5.4%	-5.5%
k=8	+4.5%	+3.9%	-3.5%

9.5 Generalizing LSI Based on NMF

Now we take a look at the classification process in a dynamic setting where new data have to be classified. Obviously, it is not suitable to compute a new NMF for every new incoming data item. Instead, a classifier is constructed by applying NMF on a training sample and using the information provided by the factors W and H in the classification model. In the following, we present adaptions of LSI based on NMF and compare them with standard LSI based on SVD. Note that in this section our datasets are transposed compared to the experiments in Sections 8.6 and 9.4. Thus, in this section, every column of A corresponds to an email message.

Review of VSM and standard LSI. A review of VSM and standard LSI based on SVD can be found in Section 4.8. In the following, LSI based on SVD is abbreviated as "SVD-LSI"

9.5.1 Two NMF-based Classifiers

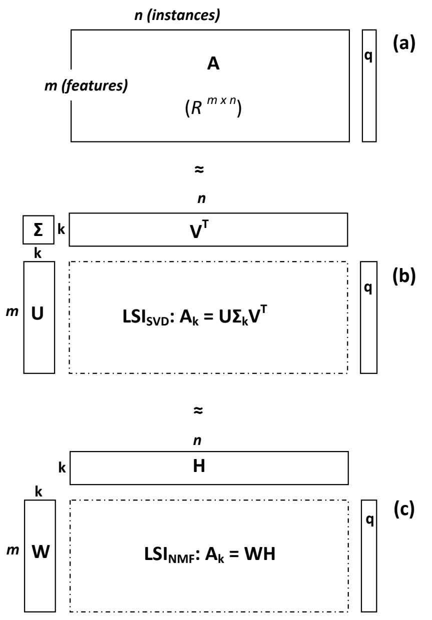

Figure 9.3: Overview – (a) basic VSM, (b) LSI using SVD, (c) LSI using NMF

We investigate two novel concepts for using NMF as low rank approximation within LSI (see Figure 9.3). The first approach which we call NMF-LSI simply replaces the approximation within LSI with a different approximation. Instead of using the truncated SVD $U_k \Sigma_k V_k^\top$, we approximate A with $A_k := W_k H_k$ from the rank k NMF. Obviously, when using NMF the value of k must be fixed prior to the computation of W and H. The cosine of the angle between $\mathbf{q}$ and the i-th column of A can then be approximated as

$$\text{(NMF-LSI)} : cos\varphi_i \approx \frac{e_i^\top H_k^\top W_k^\top q}{||W_k H_k e_i||_2 ||q||_2} \tag{9.1}$$

To save computational cost, the left term in the denominator ($e_i^\top H_k^\top$) and the left part of the enumerator ($||W_k H_k e_i||_2$) can be computed a priori.

The second new classifier which we call NMF-BCC (NMF basis coefficient classifier) is based on the idea that the basis coefficients in H can be used to classify new email. These coefficients are representations of the columns of A in the basis given by the columns of W. If W, H and $\mathbf{q}$ are given, we can calculate a column vector $\mathbf{x}$, that minimizes the equation

$$\min_x ||Wx - q||_2 \tag{9.2}$$

Since $\mathbf{x}$ is the best representation of $\mathbf{q}$ in the basis given by W, we then search for the column of H which is closest to $\mathbf{x}$ in order to assign $\mathbf{q}$ to one of the three classes. We determine the distance of $\mathbf{x}$ to the columns of H just like in VSM. The cosine of the angle between $\mathbf{x}$ (and thus of $\mathbf{q}$) and the i-th column of H can be approximated as

$$\text{(NMF-BCC)} : cos\varphi_i \approx \frac{e_i^\top H^\top x}{||He_i||_2 ||x||_2} \tag{9.3}$$

It is obvious that the computation of the cosines in Equation (9.3) is much faster than for both SVD-LSI and NMF-LSI (since usually H is a much smaller matrix than A), but the computation of $\mathbf{x}$ in Equation (9.2) causes additional cost. These aspects will be discussed further at the end of this section.

9.5.2 Classification Results

A comparison of the results achieved with LSI based on SVD (SVD-LSI), LSI based on NMF (NMF-LSI), the basis coefficient classifier (NMF-BCC) and a basic VSM is shown in Figures 9.4 and 9.5, again for different values of *maxiter*. In contrast to Section 9.4, where we performed a cross validation on the bigger email corpus, here we used the big corpus as training set and tested with the smaller corpus consisting of the 1 000 *newest* email messages of each class. For classification, we considered the column of A with the smallest angle to $\mathbf{q}$ to assign $\mathbf{q}$ to one of the classes ham, spam and phishing. The results shown in this section are based on random NMF initialization. Interestingly, the classification accuracy does not increase when using the initialization strategies mentioned in Chapter 8 (not shown in the figures).

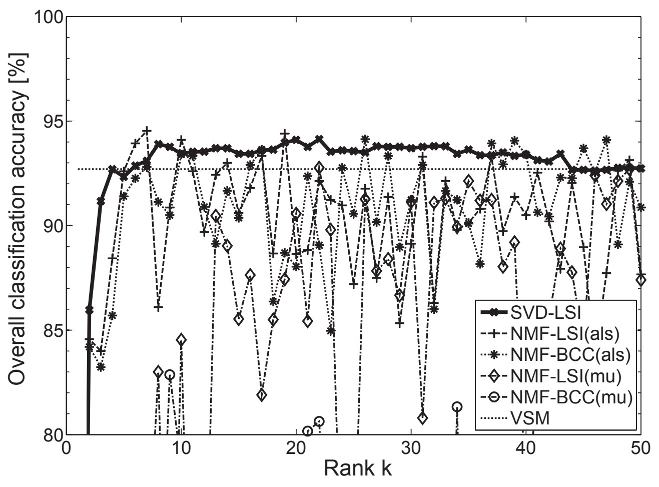

Figure 9.4: Overall classification accuracy for different LSI variants and VSM (*maxiter*=5).

Obviously, there is a big difference in the classification accuracy achieved with the NMF approaches for small and larger values of *maxiter*. With *maxiter*=5 (see Figure 9.4), the NMF variants can hardly compete with LSI based on SVD and VSM. However, when *maxiter* is increased to 30 all NMF variants except NMF-BCC(mu) show comparable results (see Figure 9.5). For many values of k, the NMF variants achieved better overall classification accuracy (cf. Section 2.4.5) than a basic VSM with all original features. Moreover, especially the standard ALS variant (NMF-LSI(als)) achieves very comparable results to LSI based on SVD, especially for small values of rank k (between 5 and 10). Note that an improvement of a few percent is substantial in the context of email classification. Moreover, as discussed in Section 8.5, the purely nonnegative linear representation within NMF makes the interpretation of the NMF factors much easier than the interpretation of the SVD factors for standard LSI. As already mentioned, an initialization of the factors W and H does not improve the classification accuracy (neither for the MU algorithm nor the ALS algorithm) when using NMF-LSI and NMF-BCC classifiers (not shown in the figures). This is in contrast to the prior sections – especially for small number of *maxiter*, the initialization was important for support vector machine.

9.5.3 Runtimes

The computational runtime for all LSI variants comprises two steps. Prior to the classification process, the low rank approximations (SVD or NMF, respectively) have to be computed. Afterwards, any newly arriving email message (a single query vector) has to be classified. For a fair comparison of runtime we used only algorithms available in the Matlab statistics toolbox (cf [Mat09b]). These algorithms include

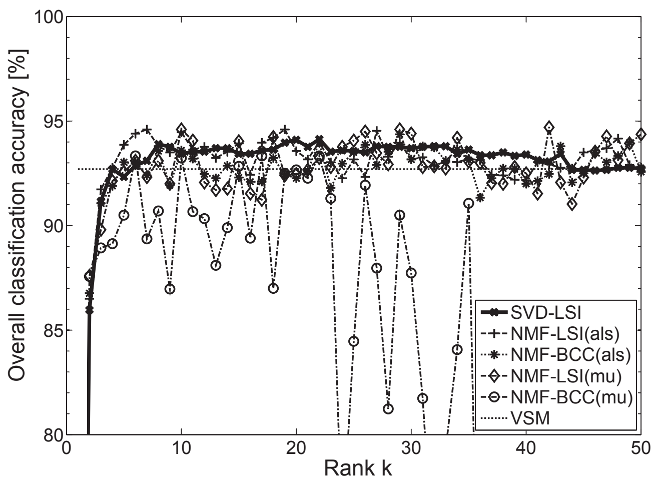

Figure 9.5: Overall classification accuracy for different LSI variants and VSM (*maxiter*=30).

the NMF algorithms MU and ALS, as well as the SVDS algorithm to compute a thin SVD (see next paragraph). For all NMF algorithms we performed the maximal number of iterations as defined by (*maxiter*) – all other convergence criteria (cf. Section 3.3.5 were switched off.

Figure 9.6 shows the runtimes needed for computing the low rank approximations, Figure 9.7 shows the runtimes for the classification process of a singly query vector. As already mentioned in Section 8.6.2, the NMF runtimes depend almost linearly on the value of *maxiter*. Figure 9.6 shows that for almost any a given rank k, the computation of an SVD takes much longer than a NMF factorization with *maxiter*=5, but is faster than a factorization with *maxiter*=30. For computing the SVD we used Matlab's `svds()` function, which computes only the first k largest singular values and associated singular vectors of a matrix. The computation of the complete SVD usually takes much longer (but is not needed in this context). There is only a small difference in the runtimes for computing the ALS algorithm (using the economy-size QR factorization, cf. Section 8.6.2) and the MU algorithm, and, of course, no difference between the NMF-LSI and the NMF-BCC runtimes (since the NMF factorization has to be computed identically for both approaches). The difference in the computational cost between NMF-LSI and NMF-BCC is embedded in the classification process of query vectors, not in the factorization process of the training data.

When looking at the classification runtimes in Figure 9.7, it can be seen that the classification process using the basis coefficients (NMF-BCC) is faster than for SVD-LSI and NMF-LSI (all algorithms were implemented in Matlab). Although

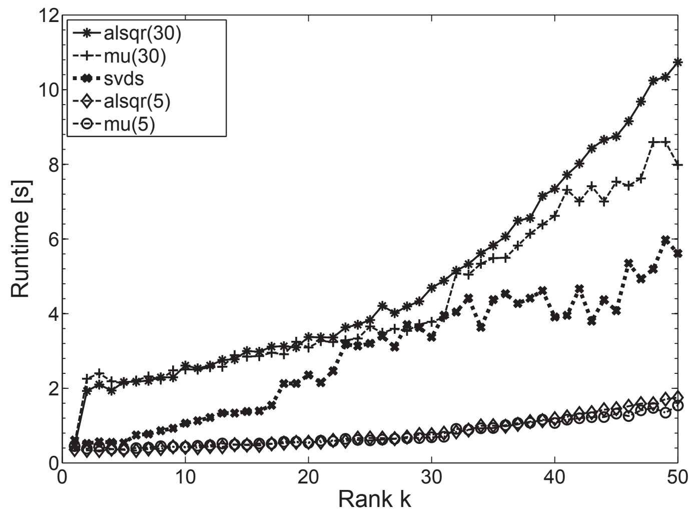

Figure 9.6: Runtimes for computing SVD and variants of NMF of a $12\,000 \times 133$ matrix. (*alsqr(30)* refers to the ALS algorithm computed with explicit QR-factorization cf. Section 8.6.2) and *maxiter* set to 30)

the classification times for a single email are modest they have to be considered for every single email that is classified. The classification (performed in MATLAB) of all 3 000 email messages in our test sample took about 36 seconds for NMF-LSI, 24 seconds for SVD-LSI and only 13 seconds for NMF-BCC (for rank k=50).

Rectangular vs. Square Data. Since the dimensions of the email data-matrix used in this work are very imbalanced ($12\,000 \times 133$), we also compared runtime and approximation errors for data of the same size, but with balanced dimensions. We created square random matrices of dimension $\sqrt{133 \times 12000} \approx 1\,263$ and performed experiments identical to the last paragraph on them.

Figure 9.8 shows the runtime needed to compute the first k largest singular values and associated singular vectors for SVD (again using the `svds()` function from MATLAB) as well as the two NMF factorizations with different values of *maxiter*. Obviously, the runtime varies significantly compared to Figure 9.6. For square A, the computation of the SVD takes much longer than for unbalanced dimensions. In contrast to that, both NMF approximations can be computed much faster (cf. Figure 9.6). For example, a computation of an SVD of rank k=50 takes about eight times as long as a computation of a NMF of the same rank.

The approximation error for square random data is shown in Figure 9.9. It is interesting to note that the approximation error of the ALS algorithm decreases with increasing k until $k \approx 35$, and then increases again with higher values of k. Nevertheless, especially for smaller values of k the ALS algorithm achieves an approximation error comparable to the SVD with much lower computational runtimes.

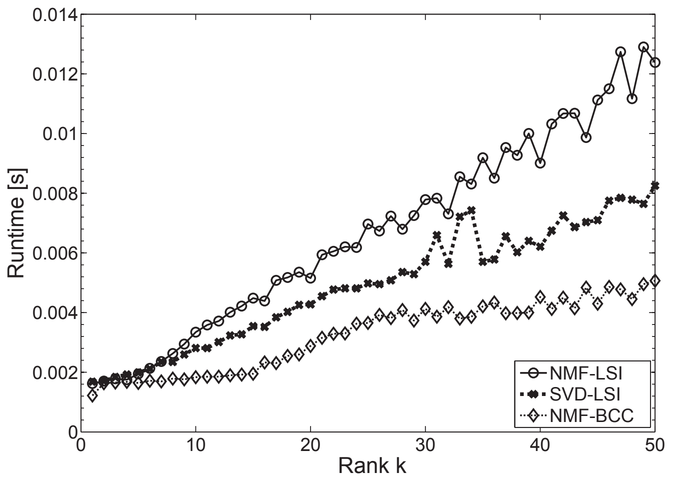

Figure 9.7: Runtimes for classifying a single query vector.

9.6 Discussion

In this chapter, the application of nonnegative matrix factorization (NMF) for supervised classification tasks has been analyzed. We investigated and evaluated two new classification methods which are based on NMF. Experimental results showed that using the basis features of W for email classification problems (same data as used in Chapter 8) generally achieves much better results than using the complete data matrix containing all original features. While the number of iterations (*maxiter*) in the iterative process for computing the NMF seems to be a crucial factor for the classification accuracy when random initialization is used, the classification results achieved with FS-initialization (as introduced in Chpater 8) and nndsvd depend only weakly on this parameter, leading to high classification accuracy even for small values of *maxiter* (see Figures 9.1 and 9.2). This is in contrast to the approximation error illustrated in Figures 8.5 and 8.6, where the number of iterations is important for all initialization variants. This indicates that a fast and accurate classification is possible for small *maxiter* and small k, even for imprecise low-rank approximations of the original data matrix.

We also introduced a new classification method based on NMF to be applied on newly arriving data without recomputing the NMF. For this purpose, we introduced two LSI classifiers based on NMF and compared them to standard LSI based on singular value decomposition (SVD). Experiments with the email dataset mentioned in Chapter 8 show that both new variants are able to achieve a classification accuracy comparable with standard LSI. Moreover, the the classification process can be performed faster, especially when the dimensions of the original data matrix are close

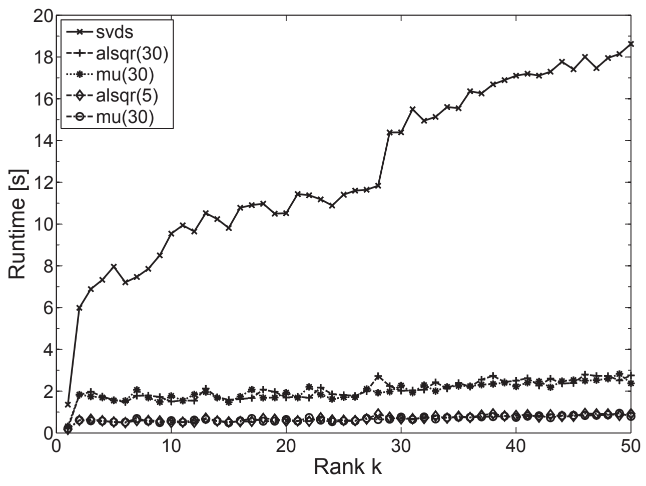

Figure 9.8: Runtimes for computing SVD and variants of NMF of a random 1263 × 1263 matrix.

to each other (in this case, the computation of the SVD usually takes much longer than a NMF factorization).

When comparing the computational cost of the LSI variants it can be seen that the runtimes depend on the structure of the data. For the imbalanced email dataset the gap between the different LSI variants is rather small (see Figures 9.6 and 9.7). For datasets with balanced dimensions there is a big difference in the computational cost for computing the low-rank approximation. In this case, all NMF variants are markedly faster than the computation of the SVD (see Figure 9.8), and thus the complete classification process can be performed faster when using NMF instead of SVD for computing the low-rank approximation within LSI.

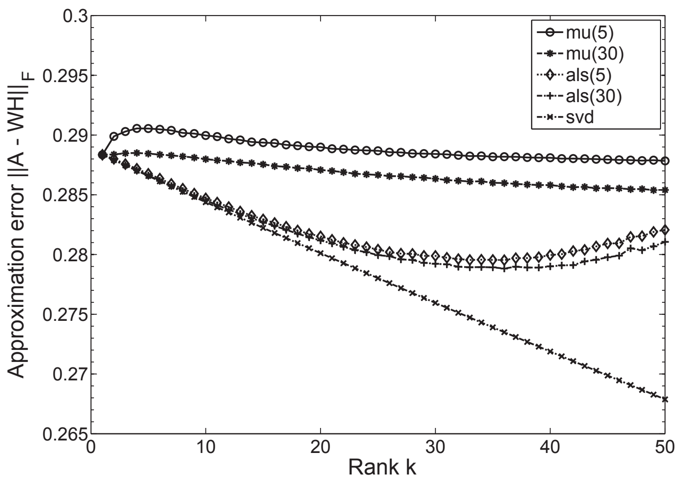

Figure 9.9: Approximation error for SVD and variants of NMF on a random 1263×1263 matrix.

Chapter 10

Improving the Performance of NMF

10.1 Overview of Chapter

The possibly huge size of the datasets used in the application areas mentioned in Chapter 5 affirms the importance of efficient implementations in these research areas. The aim of this chapter is to investigate and compare several algorithmic variants for efficiently computing nonnegative matrix factorization (NMF) on randomly created data and on data coming from in silico drug discovery problems on multi-core systems. By comparing the new variants with existing implementations, we illustrate the complex interaction between problem size, initialization strategy, accuracy of the factorization and computational efficiency on multi-core systems. For some scenarios new algorithmic variants clearly outperform existing implementations, in some cases achieving almost linear speedup on multi-core systems.

10.2 Introduction and Related Work

The goals of parallel computing are to solve given problems in less time and/or to solve bigger problems (cf. [RRB08]). Generally, parallel computing requires that a problem can be decomposed into sub-problems that can be solved independently. For performing NMF computations in parallel there are thus two possibilities: On the one hand, repetitions of the entire factorization process can be distributed and computed in parallel (*task parallelism*, for example in combination with random initialization of the factors W and H). In this case, only little effort is required to separate the problem into a number of parallel tasks. These kinds of parallel problems are also called *embarrassingly parallel* problems. On the other hand, the computations within a single factorization can be computed in parallel, for example, when performing operations in a multithreaded fashion (*data parallelism*).

Task parallelism. Task parallel computation is useful in situations where many loop iterations of independent calculations, such as a Monte Carlo simulation, need to be performed. NMF algorithms with randomly seeded factors W and H are usually repeated several times with newly randomized W and H to avoid the algorithms to get stuck in local minima (cf. [BG08]). This repetition of NMF is represented in Algorithm 1 in Section 3.3.5 in line 2 (*maxrepetition*).

Data parallelism. For NMF, data parallel computation is especially important when performing only one repetition of the factorization, for example, when working with pre-initialized factors (initialization strategies for NMF are discussed in detail in Chapter 8). Single factorizations can be computed more efficiently by spanning the computation of matrix operations (such as matrix-matrix multiplication) over multiple computational threads. On multi-core or multiprocessor systems, all threads will generally run at the same time, with each processor or core running a particular thread, thus decreasing the runtime for these complex operations.

Related work. One of the first studies that focused on parallel implementations of NMF algorithms was published in [RM06]. The authors proposed a parallel implementation of the MU algorithm (cf. Section 3.3.5) written in Java applied on hyperspectral images. Their algorithm – called parallel NMF (PNMF) – divides and distributes the NMF factors W and H among the processors. Empirical results on a multi-core machine with four processors show a good speedup when the number of threads is set to four or less. In a subsequent study [RM07] the same authors proposed a parallel implementation of Lin's projected gradient method (cf. [Lin07]) called parallel adaptive projected NMF (PAPNMF). This algorithm seeks the best step size parameter α concurrently, thus improving the runtime of the algorithm. The results of the papers [RM06, RM07] are summarized in a journal article in [RM09].

An unpublished article [Kan07] presents parallel NMF implementations based on openMP (cf. [OMP09]) for document clustering. The authors mention that their parallel implementations of the three NMF algorithms presented in [LS99, XLG03, DLP06] achieve a good speedup and efficiency when the number of clusters is much smaller than both the number of words/tokens extracted from the documents *and* the number of documents.

[MCN08] have developed a web-based tool called bioNMF that implemented the NMF methodology in different analysis contexts to support applications in biology (including clustering and biclustering of gene-expression data and sample classification). The NMF factors are broken into sub-matrices, and operations between these sub-matrices are computed in parallel. The communication of their parallel implementation is based on the message passing interface (MPI, cf. [MPI09]). Since the authors compared their algorithm with a Matlab implementation of NMF proposed in [BTG04], the reported speedup of almost seven might be overrated. Another MPI-based parallel NMF implementation for large-scale biological data can be found in [DW08].

In [ZBL09] the authors proposed a parallel implementation for nonnegative tensor factorization (NTF), a natural extension of NMF to higher dimensional data. This study exploits the NTF for multidimensional climate data. The results show a sublinear speedup over a sequential execution when using 2 to 8 processors on a multi-core machine with an approximate peak speedup of 6.8 (among all runs with up to 10 processors). The authors mention that their results were achieved with the original Matlab NTF code rewritten in C++ compiled with several libraries including LAPACK and ScaLAPACK (cf. Section 3.3.3).

A very recent study [DZW09] focused on parallel NMF implementations to be used explicitly on distributed memory platforms with MPI. The authors proposed an adaption of the algorithms presented in [RM06, RM07] to distributed memory systems. To eliminate the network consumption, a new algorithm was proposed which divides the dataset into uncorrelated sub-blocks that are processed by different nodes simultaneously. Experiments demonstrate that the proposed methods are acceptable in both precision and efficiency.

10.3 Open Issues and Own Contributions

Many existing parallel NMF implementations achieve good speedup and high efficiency, but most of them are written in low-level parallel code and thus require (at least) major modifications of the program code and the utilization of a communication protocol such as MPI (cf. [MPI09]), openMP (cf. [OMP09]), or a similar communication protocol. Several difficulties arise when programming low-level parallel code. For example, communication and synchronization between the different subtasks has to be handled. This increases the risk of potential software bugs, such as race conditions or deadlocks. Moreover, programmers have to pay attention to hardware and network architectures. Low-level parallel code is also harder to write, understand, modify and debug than sequential code. Many problems that can be solved in a couple of dozen lines of sequential code require hundreds or sometimes thousands of lines of code to be solved efficiently in parallel.

The goal of this chapter is to investigate new algorithmic variants which may lead to better and more scalable parallelization, ideally without the restrictions and problems of low-level parallel programming. One possibility to achieve this goal is to use high-level programming language libraries that simplify parallel code development by abstracting away the complexity of managing coordination and distribution of computations and data between several processors. The Matlab parallel computing toolbox (cf. [Mat09a]) is an example of such a high-level library that provides several high-level programming constructs for converting serial Matlab code to run in parallel. The parallel processing constructs implemented in the Matlab parallel computing toolbox allow for task- and data-parallel algorithms without forcing the programmer to take care of specific hardware and network architectures. As a result, converting serial Matlab programs to parallel Matlab programs requires only few code modifications and no programming in a low-level language.

In this chapter, we utilize and adapt various algorithmic variants based on Matlab to exploit task- and data-parallelism for NMF. Based on these comparisons, the complex interaction between problem size, initialization strategy, accuracy of the factorization and computational efficiency of NMF computations on multi-core systems is illustrated. Moreover, we introduce a computationally very efficient adaption for computing Matlab's implementation of the ALS algorithm without any noticeable disadvantages in the drug discovery context. This algorithm also provides the several possibilities to be executed parallel.

10.4 Hardware, Software, Datasets

In the following, we briefly describe the three different groups of data used for the experiments in this chapter, as well as the hardware and software architecture used.

10.4.1 Datasets

The experimental results in this chapter are based on various datasets that can be clustered into three groups. For results in Section 10.7 we used the drug discovery dataset (*dd_big*), for computing runtimes and computational efficiency of the algorithms in Sections 10.5 and 10.6 we used randomly created matrices of various sizes. The three groups of datasets are summarized briefly in the following list:

- Drug discovery dataset (*dd_big*): This dataset contains 122 000 chemical compounds that are described by 268 different chemical descriptors. 200 of these compounds are known to have a specific biological activity, the remaining majority do not have this biological activity. About 10% of all entries have zero values.

- Random small (*rand_small*): In this group of datasets, the dense data matrices A are square and have a size from $m, n = 250$ to $m, n = 2\,500$ (with a step-size of 250). The nonnegative factors W and H were also randomly seeded with a fixed $k = 50$.

- Random big (*rand_big*): Similar to the small random dataset, the dense data matrices A are square. Here, their size ranges from m, n=1 000 to m, n=7 000 (with a step-size of 1 000). Contrary to the small datasets, the factors W and H were (also randomly) seeded with different values of k ranging from 50, 150, 250, 500 to 75% of m, n (i. e., k=750 for m, n=1 000).

10.4.2 Hardware Architecture

The experiments presented in this chapter were performed a SUN Fire X4600M2 with 8 AMD quad-core Opteron 8356 processors (32 cores overall) with 2.3GHz CPU and 64GB of memory running under 64-bit Linux Ubuntu 8.10 (kernel 2.6.27-11).

10.4.3 Software Architecture

We used Matlab-based implementations of NMF algorithms to investigate the performance of task-parallel NMF variants as well as the performance difference between single-threaded and implicit multithreaded factorization (without any code changes). In the following, we briefly summarize Matlab's NMF implementation and its multithreading possibilities. We also describe how the Matlab NMF implementation can be adapted to perform task-parallel NMF.

Computing NMF in Matlab. Matlab provides a built-in function for computing NMF with two of the algorithms described in Section 3.3.5, the multiplicative update algorithm (MU) and the alternating least squares algorithm (ALS). This function (called *nnmf.m*) has been included in the statistics toolbox since version 6.2 (R2008a). For experimental evaluation in this paper we used the 64-bit variants of Matlab versions R2008b and R2009a.

Multithreading. Matlab allows for implicit multithreading since multithreaded computations are supported for a number of linear algebra operations (e. g., matrix-matrix multiplication). These functions automatically execute on multiple threads without explicitly specifying commands to create threads in the code. By default, the number of threads used by Matlab is set to the number of cores that are available on a machine. To be able to compare single- and multithreaded runtimes, we partly switched off the multithreading capabilities and executed jobs sequentially on one thread (i. e., only one core was used). Until Matlab 2008b, the number of computational threads to be used could be adjusted by the user. Since Matlab 2009a, the capability to adjust the number of threads in a single Matlab session is no longer available. Matlab argues that "the primary reason for this change is that products that Matlab is dependent upon have the ability to spawn threads from the currently-executing thread. This makes it infeasible to monitor and/or limit the number of computational threads at any given time in a Matlab process." [Mat09b].

Task parallelism. Recent Matlab versions support task parallel processing opportunities on multi-core and multiprocessor systems. As mentioned in Chapter 8, NMF algorithms with random initializations are usually repeated several times, which makes NMF suitable for task parallel computation. A very simple but fast and efficient way to perform task parallel computation in Matlab proceeds as follows: Prior to the distribution of parallel tasks, a set of Matlab workers (also called *labs*) has to be defined. Like threads, workers are executed on processor cores (the number of workers does not have to match the number of cores), but unlike threads, workers do *not* share memory with each other. The workers can now either be used to offload work to them (for example by sending a batch job to a worker) or to execute parallel for-loops (called *parfor*-loops, cf. [Mat09a]). Each *parfor*-loop divides the loop iterations into groups so that each worker executes some portion of the total number of iterations. Part of the *parfor* body is executed on the Matlab client (where the *parfor* is issued) and part is executed in parallel on Matlab workers. The necessary data on which *parfor* operates is sent from the client to workers, where most of the computation happens, and the results are sent back to the client and pieced together (for details see Matlab's documentation of the parallel computing toolbox, cf. [Mat09a]).

Task parallel computations in Matlab are supported by two products: the *parallel computing toolbox* and the *distributed server*. The parallel computing toolbox version 4.1 (part of Matlab 2009a) supports applications to run on up to eight local workers. In the preceding version of the parallel computing toolbox (version 4.0, part of Matlab 2008b), the number of workers was limited to four. The Matlab distributed server offers the possibility to run code on workers that might be running remotely on a cluster and do not need to run on the local machine. However, for the experiments summarized in this chapter we did not have Matlab's distributed server available. The results discussed here were based on the parallel computing toolbox.

10.5 Task-Parallel Speedup

In this section, we investigate the speedup when using task-parallel NMF algorithms. To achieve this task-parallelism, we utilized Matlab's parallel computing toolbox' possibility to distribute tasks across multiple Matlab workers using *parfor*-loops. As mentioned before, Matlab 2008b (parallel computing toolbox 4.0) supports up to four workers, Matlab 2009a (parallel computing toolbox 4.1) supports up to eight workers. For evaluation in this section, we set the number of iterations (*maxiter*) to 100 (all other termination criteria mentioned in Section 3.3.5 where switched off), and the number of repetitions (*maxrepetitions*) to 50 (see Algorithm 1 in Section 3.3.5). All results are based on the small randomized datasets *rand_small* (see Section 10.4.1).

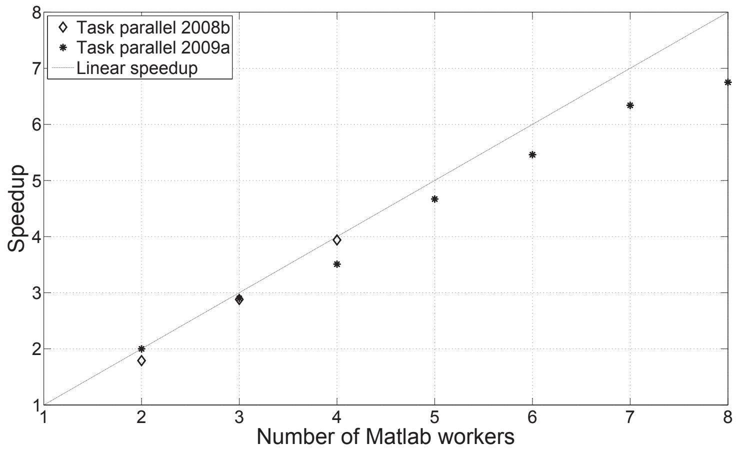

Figure 10.1: Task-parallel speedup for MU algorithm. Baseline: Single Matlab worker.

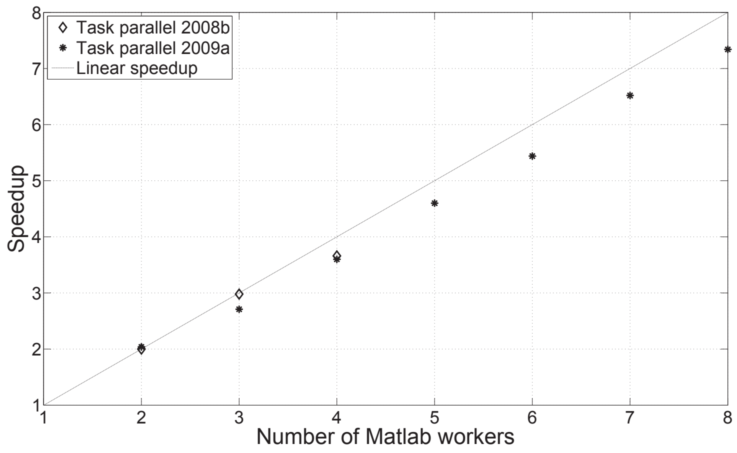

Figure 10.2: Task-parallel speedup for ALS algorithm. Baseline: Single Matlab worker.

The repetitions of the NMF algorithms are distributed over the Matlab workers and are computed in parallel. Prior to this job-distribution a pool of Matlab workers needs to be created (see Section 10.4.3). Figures 10.1 and 10.2 show the runtime achieved with the MU and the ALS algorithm, respectively. The speedups shown in this section are the average results over all datasets included in *rand_small* (see Section 10.4.1). Here, Matlab's possibility to use thread parallelism was turned off,

i. e., the results with only one Matlab worker were computed on one core. For each additional worker, one additional core was used.

Results. The results on our SUN Fire (Figures 10.1 and 10.2) show very stable speedup for both algorithms, MU and ALS. As can be seen, with increasing number of workers the speedup increases almost linearly. The difference between the achieved results and the (theoretical) linear speedup is caused by the overhead resulting from creating Matlab workers, assigning jobs and data to them, and communication between workers and the Matlab client.

10.6 Improvements for Single Factorizations

As discussed in Chapter 8, NMF algorithms strongly depend on the initialization of W and H. Some non-random initialization strategies have shown to improve NMF in terms of faster convergence and faster error reduction [BG08, JG09]. In the case of non-random initialization only a single factorization has to be performed (since the results will not change). This raises the importance to speed up the runtime for a single factorization instead of parallelizing several NMF repetitions over several machines or cores.

One possibility to speed up a single NMF factorization is to compute linear algebra operations (such as matrix-matrix multiplication) over multiple computational threads. As mentioned before in Section 10.2, all threads will generally run at the same time, with each processor or core running a particular thread. Another possibility is to reduce the computational effort in each iteration. In Section 10.6.2 we introduce a computationally very efficient adaption for computing Matlab's ALS implementation.

10.6.1 Multithreading Improvements

Table 10.1 shows the speedup achieved with different Matlab variants using the datasets *rand_small* (cf. Section 10.4.1). The single threaded Matlab version 2008b was used as a baseline. For computing the runtimes of multithreaded Matlab versions, the number of threads was set to the number of cores available on the system (32 in our case). Some remarks on Matlab's multithreading possibilities are given in Section 10.4.3.

Results. As can be seen, no speedup can be achieved when using multithreaded Matlab 2008b. Although the number of threads to be used was set to the number of cores of each machine, computations were only performed on one core. This behavior was also noticed on other hardware architectures. The multithreaded Matlab 2009a shows a speedup of almost 2 with similar results for both algorithms (MU and ALS).

Matlab version	**MU**	**ALS**
2008b single thread	1.00	1.00
2008b multithreaded	0.99	0.97
2009a single thread	1.02	1.00
2009a multithreaded	1.93	1.94

Table 10.1: NMF speedups for different Matlab's versions using datasets *rand_small*. Baseline: Matlab 2008b single thread

10.6.2 Improving Matlab's ALS Code

We also investigated several variants of the ALS algorithm. In particular, in the following we illustrate how the computational efficiency of Matlab's implementation of the ALS algorithm can be improved significantly. In Algorithm 9 we depict the central steps of Matlab's implementation (using Matlab notation) of the ALS algorithm (cf. Algorithm 3 in Section 3.3.5).

Algorithm 9 - Matlab ALS update step.

```
1: h = max(0, w0\a);
2: w = max(0, a/h);
```

According to the Matlab convention, the "\"-operator (left matrix division) in Algorithm 9 (line one) is to be interpreted as follows: If A is an m-by-n matrix with m $\neq$ n and B is a column vector with m components, or a matrix with several such columns, then $X = A\backslash B$ is the solution in the least squares sense to the potentially underdetermined or overdetermined system of equations $AX = B$ (i. e., X minimizes the norm$(A*X-B)$). The "/"-operator (right matrix division) in Algorithm 9 (line two) is a compact notation for the left matrix division $(A^\top\backslash B^\top)^\top$.

As an alternative, we experimented with the normal equations formulation of the ALS algorithm (cf. Algorithm 3, Section 3.3.4), denoted as NEALS in the following. The Matlab formulation of the NEALS algorithm is illustrated in Algorithm 10.

Algorithm 10 - NEALS update step.

```
1: temp1 = w0' × w0;
2: temp2 = w0' × a;
3: h = max(0, temp1\temp2);
4:
5: temp3 = h × h';
6: temp4 = h × a';
7: w = max(0, temp3\temp4)';
```

Although it is well known that in general this formulation has numerical disadvantages because of the squaring of the condition numbers of the factors W and

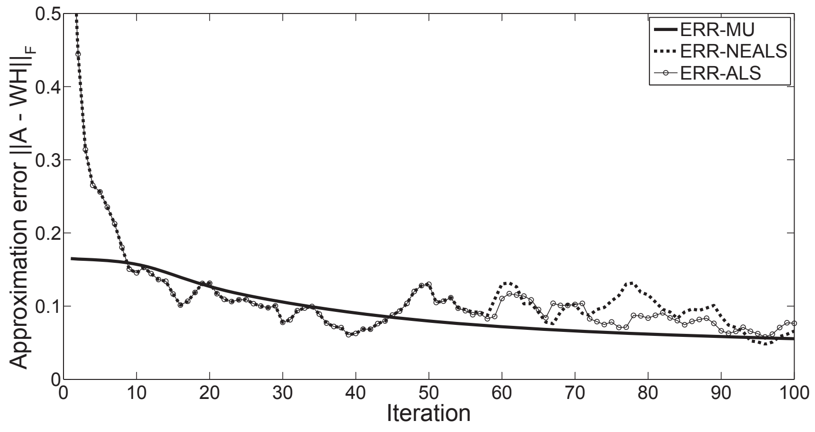

Figure 10.3: Convergence history for MU, NEALS and ALS algorithm.

H, in the given context it can be very attractive for various reasons: (i) Since in many cases we need to compute a *low-rank* NMF with relatively small k, the resulting normal equations are much smaller than the least squares formulations in Algorithm 9. (ii) The potential loss in numerical accuracy is usually not too severe, because the computation of an NMF is only an approximation anyway. (iii) Obviously, the additional expense is the matrix multiplication required for forming the normal equations. However, it is well known that this operation has a favorable computation per communication ratio and thus can be mapped well onto multi-core architectures.

As Figure 10.3 illustrates, the convergence properties of the ALS algorithm are usually not dominated by the aspect of which actual algorithmic variant to be used. As can be seen, for this selected example ALS and NEALS show identical results until about 60 iterations. In about 95% of our test-runs, NEALS and ALS achived numerically identical results for all iterations (always set to $maxiter = 100$).

The considerations mentioned before are confirmed experimentally on dataset *rand_big* (cf. Section 10.4.1), as Figure 10.4 shows. The normal equations formulation achieves high speedups compared to the standard implementation provided in Matlab. There results were achieved with the multitreaded Matlab 2009a version.

Parallel NEALS. Looking at lines 1 and 2 as well as lines 3 and 4 of Algorithm 10 it can be seen that the computation of the help-variables temp$x$ can be computed independently. However, with the means currently available to us we were not able to fully utilize the parallelization potential in the NEALS formulation in terms of task parallelism. Investigations of this parallelization potential will be part of our future work.

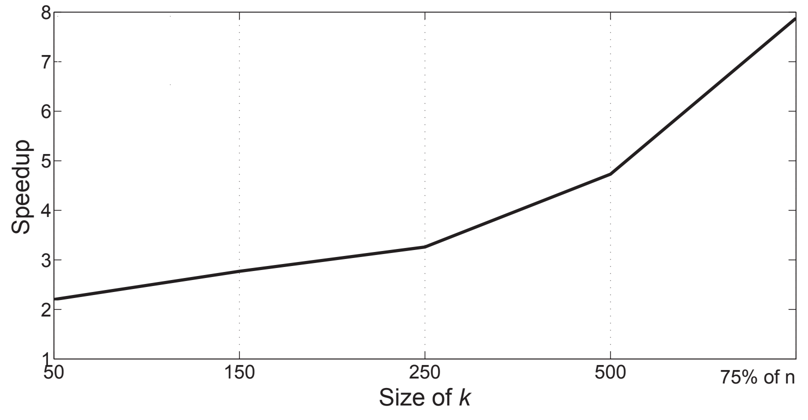

Figure 10.4: Speedup of NEALS algorithm over standard Matlab ALS algorithm.

10.7 Initialization vs. Task-parallelism

In this section, we apply the initialization techniques introduced in Chapter 8 to our drug discovery dataset *dd_big* (cf. Section 10.4.1). NMF factorizations achieved with random initialization (using task-parallel NMF algorithms), and results achieved with information gain initialization (using data-parallel NMF algorithms) are compared, and the interaction between the initialization strategy, the accuracy of the factorization achieved with different algorithms and their computational efficiency is discussed.

The results based on random initialization were achieved using task-parallel implementations mentioned in Section 10.5. Here, the number of repetitions of the complete factorization (*maxrepetitions*) was set to the number of Matlab workers available. Since the results in this section were achieved using Matlab 2009a, eight workers were available. Information gain results were achieved exploiting Matlab's data-parallel potential mentioned in Section 10.6. Here, the number of threads was again set to the number of cores available on the system (32 in our case). All results in this section were achieved with k=10, and the number of iterations (*maxiter*) was set to 50 (all other termination criteria mentioned in Section 3.3.5 where switched off).

Figure 10.5 shows the NMF results for the three algorithms MU, ALS and NEALS for the *dd_big* file when using (*i*) random initialization ("ERR-rand") and (*ii*) information gain-based initialization ("ERR-IG") as described in Chapter 8. The random initialization results shown in this figure are the *average* results of eight task-parallel factorizations (using different initializations of W and H for each repetition), i. e., the expected results using randomized initialization. Obviously, the IG initialized factorization achieves a better accuracy (i. e., a smaller NMF error as

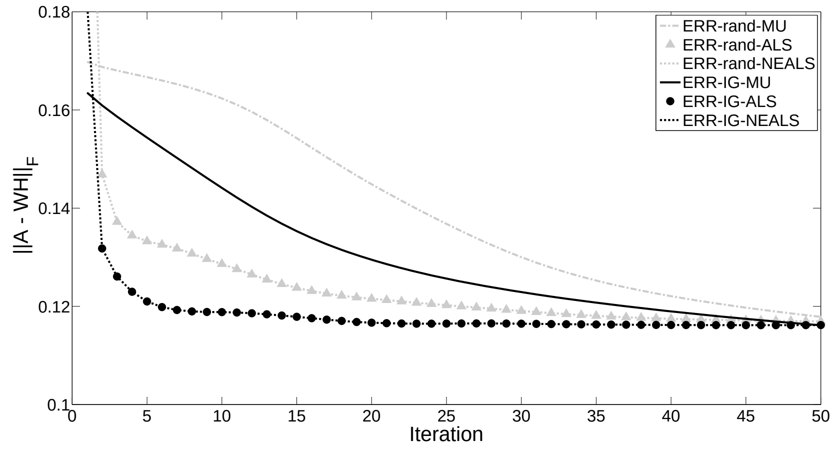

Figure 10.5: IG initialization vs. average result of eight random initializations.

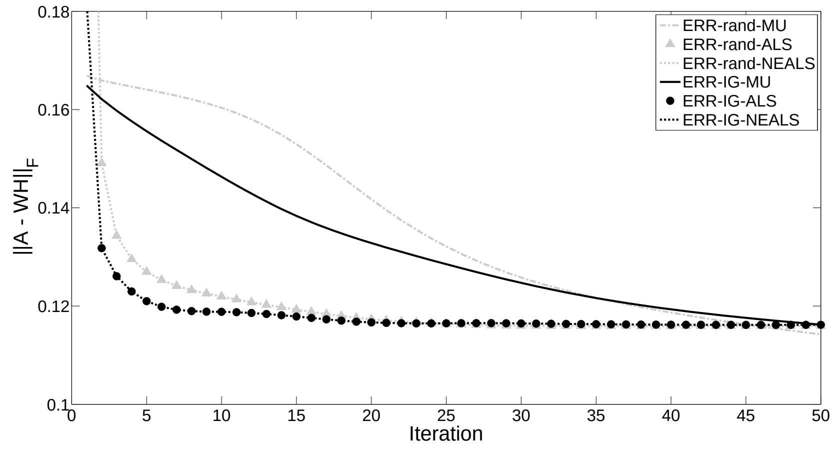

Figure 10.6: IG initialization vs. best result of eight random initialization.

stated in Equation (3.17)) than randomly initialized NMF. Especially when the number of iterations (*maxiter*) is small, the IG results are much better than the results achieved with random initialization. When comparing the different NMF algorithms used, it can be seen that a reduction of the approximation error can be achieved with fewer iterations when using the ALS (and NEALS) algorithm compared of the MU algorithm. However, with increasing number of iterations the MU algorithm shows a continuous decrease in the error, while ALS and NEALS sometimes tend to converge to local minima and are unable to further reduce the factorization error (cf., for example, [SRG03]).

Figure 10.6 shows the results for the same algorithms as Figure 10.5, but this time the random initialization results are shown for the *best* task-parallel factorization, i. e., the factorization with the smallest approximation error after 50 iterations. Obviously, the approximation error achieved with the best random initialization is better than the average results shown in Figure 10.5. Nevertheless, the results achieved with information gain initialization are still better for a small number of iterations. When using the MU algorithm, the IG results are better until about 35 iterations. For a larger number of iterations, the best randomly initialized NMF achieves slightly better results. When using the ALS or the NEALS algorithm, IG initialization is better until 23 iterations – for larger numbers the results are almost identical.

Runtimes. Table 10.2 shows the approximation error for IG initialization after a given number of iterations and the runtime needed to perform these numbers of iterations ("ERR-IG" results in Figure 10.6). The computational cost for performing the IG initialization is *not* included in these runtimes. As already mentioned, the IG runtimes were achieved exploiting Matlab's data-parallel potential. As can be seen, the ALS algorithm is the slowest algorithm in terms of runtime per iteration compared to MU and NEALS. Overall it can be seen that NEALS is the fastest algorithm and that ALS and NEALS need significantly fewer iterations than MU to reduce the approximation error for a small number of iterations (see also Figuress 10.5 and 10.6). For example, NEALS needs only 5 iterations to achieve the same (or even better) accuracy than MU after 25 iterations (0.1268). In this case, the runtime needed to achieve this accuracy can be decreased by a factor of five when using NEALS (5.50s vs. 27.37s) instead of MU. Compared to ALS, NEALS is almost three times faster. With increasing rank k, the speedup of NEALS over the standard ALS implementation of Matlab is expected to grow even larger (cf. Figure 10.4.

Table 10.3 also shows approximation errors and corresponding runtimes, here for a task-parallel NMF using random initialization ("ERR-rand" results in Figure 10.6). NEALS is again the fastest algorithm (both in terms of runtime per iteration and in terms of runtime needed to achieve a low approximation error). When comparing the runtimes with Table 10.2 it can be seen that the computational cost needed for performing eight task-parallel NMF computations in parallel is higher than for a single data-parallel NMF. Generally, the runtime for computing eight task-parallel NMF factorizations increases by a factor of 2.2 compared to a single data-parallel factorization. This indicates that a low approximation error can be achieved faster when using pre-initialized NMF factors instead of several randomly initialized factorizations. However, as already mentioned, the runtimes in Table 10.2 do not include the computational cost for computing the initialization of W and H. In cases, where these cost might be very high, many task-parallel factorizations using random initialization might be faster. Moreover, several initialization variants (such as the IG or gain ratio initialization, cf. Chapter 8) cannot applied when unlabeled data is used (i. e., the class label ob items is unknown).

Iterations (MU)	**5**	**10**	**25**	**50**
Runtime (sec.)	6.08	11.29	27.37	53.51
Accuracy achieved	0.1553	0.1464	0.1268	0.1171
Iterations (ALS)	**5**	**10**	**25**	**50**
Runtime (sec.)	15.13	29.23	70.14	143.48
Accuracy achieved	0.1212	0.1188	0.1165	0.1162
Iterations (NEALS)	**5**	**10**	**25**	**50**
Runtime (sec.)	5.50	10.91	25.14	50.35
Accuracy achieved	0.1212	0.1188	0.1165	0.1162

Table 10.2: MU vs. ALS vs. NEALS concerning number of iterations, accuracy and runtime for IG initialization (data-parallelism).

Iterations (MU)	**5**	**10**	**25**	**50**
Runtime (sec.)	13.88	25.76	62.47	122.12
Accuracy achieved	0.1671	0.1641	0.1363	0.1157
Iterations (ALS)	**5**	**10**	**25**	**50**
Runtime (sec.)	34.53	66.71	160.09	327.47
Accuracy achieved	0.1275	0.1229	0.1181	0.1169
Iterations (NEALS)	**5**	**10**	**25**	**50**
Runtime (sec.)	12.56	24.89	57.38	121.04
Accuracy achieved	0.1275	0.1229	0.1181	0.1169

Table 10.3: MU vs. ALS vs. NEALS concerning number of iterations, accuracy and runtime for best random initialization (task-parallelism).

10.8 Discussion

Several algorithmic variants for efficiently computing the nonnegative matrix factorization (NMF) on multi-core architectures have been investigated and applied on randomly created data as well as data coming from in silico drug discovery problems. In this application context, feature reduction and interpretation of the reduced feature sets is crucial, and thus the NMF is an interesting concept for such problems. In particular, we investigated how to exploit task and data parallelism in NMF computations on the basis of Matlab codes using Matlab's parallel computing toolbox. Moreover, a modification of Matlab's implementation of the ALS algorithm has been investigated which achieves much higher performance in the given problem context.

All these different variants of NMF computations have been compared in terms of execution times and speedup on multi-core systems in order to illustrate the complex interaction between problem size, initialization strategy, accuracy of the factorization and computational efficiency. In particular, it has been shown that random initialization strategies provide a high potential for task parallelism, and almost linear speedup in the number of cores used could be achieved. The modification of Matlab's ALS algorithm based on the normal equations was shown to be up to eight times faster than the standard Matlab implementation without significant numerical disadvantages. Even higher performance gains seem possible, since its full parallel potential has not yet been exploited.

Chapter 11

Conclusion and Future Work

As the data grows, several data mining techniques become significantly harder and the computational effort for data mining applications increases dramatically. Moreover, the increasing dimensionality of data (in the sense of many features) for various application areas (such as email classification and in silico screening for drug discovery) often leads to severe problems for learning algorithms, since the high dimensional nature of data can have a negative influence on the classification accuracy and also increases the computational cost for both, supervised and unsupervised learning methods. High dimensionality of data in the sense of many instances also increases the computational cost for the learning algorithm but has usually a positive influence on the classification accuracy. This causes the need for effective and efficient feature selection and dimensionality reduction methods. Generally, these methods should meet the following demands: They should be *computationally efficient* and the resulting feature sets should allow for a *compact* representation of the original data. Moreover, in many application areas it is very important not to loose the *interpretability* of the original features. Finally, the feature reduction process should have a *positive* (or at least no negative) *effect* on the classification accuracy of the learning algorithm.

The major contributions of this thesis include (*i*) new efficient initialization strategies for nonnegative matrix factorization (NMF) which lead to very fast reduction in approximation error and can be applied in combination with (*ii*) new and very effective classification algorithms which utilize the nonnegative factors of NMF for developing a classification model. To further speed up the runtime of NMF we have (*iii*) investigated and compared several algorithmic variants for efficiently computing NMF on multi-core systems, and have (*iv*) presented a computationally very efficient adaption for Matlab's implementation of the ALS algorithm. Moreover, we have (*v*) investigated the application of latent semantic indexing (LSI) to the task of email categorization and (*vi*) have presented a comprehensive empirical study on the relationship between various methods for feature reduction (feature subset selection as well as dimensionality reduction) and the resulting classification performance.

The initialization strategies for NMF presented in this thesis allow for a faster reduction in approximation error, thus reducing the runtime needed to achieve a certain approximation accuracy. Moreover, the task- and data-parallel algorithmic variants for efficiently computing NMF on multi-core architectures, and the computationally very efficient adaptation of Matlab's ALS algorithm (Matlab version 2009a) presented in this thesis are further contributions to decrease the computational cost of NMF.

Dimensionality reduction techniques allow for compact representations of the data without losing a lot of information of the original attribute space. However, the resulting linear combinations are usually hard to interpret since they are based on additive *and* subtractive combinations of the original features. Due to the "sum-of-parts" representation of NMF, the interpretability of how much an original attribute contributes to the basis vectors and basis coefficients in NMF can be retained. In this thesis, we utilize this key characteristic of NMF. By taking advantage of the information provided in the basic vectors of NMF, important and well discriminating features can be identified in an unsupervised manner.

Moreover, we have presented two new fast and efficient classification methods based on NMF. The first method utilizes the nonnegative factors of the NMF for classification, and generally achieves much better results than using the complete data matrix containing all original features. As alternative methods, we introduced two latent semantic indexing (LSI) classifiers based on NMF. Both new variants are able to achieve classification accuracy comparable with standard LSI, but can be computed much faster, especially when the dimensions of the original data matrix are close to each other.

The strong influence of different feature reduction methods on the classification accuracy observed underlines the need for further investigation in the complex interaction between features reduction and classification. Based on the results of this thesis, the following directions for future research are identified, such as the initialization of the basis coefficient matrix in NMF, a detailed analysis of the computational cost of various initialization strategies, and stronger utilization of sparseness constraints for NMF.

Zusammenfassung

Durch die steigende Anzahl verfügbarer Daten in unterschiedlichsten Anwendungsgebieten nimmt der Aufwand vieler Data-Mining Applikationen signifikant zu. Speziell hochdimensionierte Daten (Daten die über viele verschiedene Attribute beschrieben werden) können ein großes Problem für viele Data-Mining Anwendungen darstellen. Neben höheren Laufzeiten können dadurch sowohl für überwachte (supervised), als auch nicht überwachte (unsupervised) Klassifikationsalgorithmen weitere Komplikationen entstehen (z.B. ungenaue Klassifikationsgenauigkeit, schlechte Clustering-Eigenschaften, ...). Dies führt zu einem Bedarf an effektiven und effizienten Methoden zur Dimensionsreduzierung. *Feature Selection* (die Auswahl eines Subsets von Originalattributen) und *Dimensionality Reduction* (Transformation von Originalattribute in (Linear)-Kombinationen der Originalattribute) sind zwei wichtige Methoden um die Dimension von Daten zu reduzieren. Obwohl sich in den letzten Jahren vielen Studien mit diesen Methoden beschäftigt haben, gibt es immer noch viele offene Fragestellungen in diesem Forschungsgebiet. Darüber hinaus ergeben sich in vielen Anwendungsbereichen durch die immer weiter steigende Anzahl an verfügbaren und verwendeten Attributen und Features laufend neue Probleme. Das Ziel dieser Dissertation ist es, verschiedene Fragenstellungen in diesem Bereich genau zu analysieren und Verbesserungsmöglichkeiten zu entwickeln.

Grundsätzlich, werden folgende Ansprüche an Methoden zur Feature Selection und Dimensionality Reduction gestellt: Die Methoden sollten effizient (bezüglich ihres Rechenaufwandes) sein und die resultierenden Feature-Sets sollten die Originaldaten möglichst kompakt repräsentieren können. Darüber hinaus ist es in vielen Anwendungsgebieten wichtig, die Interpretierbarkeit der Originaldaten beizubehalten. Letztendlich sollte der Prozess der Dimensionsreduzierung keinen negativen Effekt auf die Klassifikationsgenauigkeit haben - sondern idealerweise, diese noch verbessern.

Offene Problemstellungen in diesem Bereich betreffen unter anderem den Zusammenhang zwischen Methoden zur Dimensionsreduzierung und der resultierenden Klassifikationsgenauigkeit, wobei sowohl eine möglichst kompakte Repräsentation der Daten, als auch eine hohe Klassifikationsgenauigkeit erzielt werden sollen. Wie bereits erwähnt, ergibt sich durch die große Anzahl an Daten auch ein erhöhter Rechenaufwand, weshalb schnelle und effektive Methoden zur Dimensionsreduzierung entwickelt werden müssen, bzw. existierende Methoden verbessert werden müssen. Darüber hinaus sollte natürlich auch der Rechenaufwand der verwendeten Klassifikationsmethoden möglichst gering sein. Des Weiteren ist die Interpretierbarkeit von Feature Sets zwar möglich, wenn Feature Selection Methoden für die Dimen-

sionsreduzierung verwendet werden, im Fall von Dimensionality Reduction sind die resultierenden Feature Sets jedoch meist Linearkombinationen der Originalfeatures. Daher ist es schwierig zu überprüfen, wie viel Information einzelne Originalfeatures beitragen.

Im Rahmen dieser Dissertation konnten wichtige Beiträge zu den oben genannten Problemstellungen präsentiert werden: Es wurden neue, effiziente Initialisierungsvarianten für die Dimensionality Reduction Methode *Nonnegative Matrix Factorization* (NMF) entwickelt, welche im Vergleich zu randomisierter Initialisierung und im Vergleich zu State-of-the-Art Initialisierungsmethoden zu einer schnelleren Reduktion des Approximationsfehlers führen. Diese Initialisierungsvarianten können darüber hinaus mit neu entwickelten und sehr effektiven Klassifikationsalgorithmen basierend auf NMF kombiniert werden. Um die Laufzeit von NMF weiter zu steigern wurden unterschiedliche Varianten von NMF Algorithmen auf Multi-Prozessor Systemen vorgestellt, welche sowohl Task- als auch Datenparallelismus unterstützen und zu einer erheblichen Reduktion der Laufzeit für NMF führen. Außerdem wurde eine effektive Verbesserung der Matlab Implementierung des ALS Algorithmus vorgestellt. Darüber hinaus wurde eine Technik aus dem Bereich des Information Retrieval – Latent Semantic Indexing – erfolgreich als Klassifikationsalgorithmus für Email Daten angewendet. Schließlich wurde eine ausführliche empirische Studie über den Zusammenhang verschiedener Feature Reduction Methoden (Feature Selection und Dimensionality Reduction) und der resultierenden Klassifikationsgenauigkeit unterschiedlicher Lernalgorithmen präsentiert.

Der starke Einfluss unterschiedlicher Methoden zur Dimensionsreduzierung auf die resultierende Klassifikationsgenauigkeit unterstreicht dass noch weitere Untersuchungen notwendig sind um das komplexe Zusammenspiel von Dimensionsreduzierung und Klassifikation genau analysieren zu können.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAR07] Mohammed Said Abual-Rub, Rosni Abdullah, and Nuraini Abdul Rashid. "A Modified Vector Space Model for Protein Retrieval." *IJCSNS International Journal of Computer Science and Network Security*, **7**(9):85–90, 2007.

[AB95] David W. Aha and Richard L. Bankert. "A Comparative Evaluation of Sequential Feature Selection Algorithms." In D. Fisher and H. Lenz, editors, *In Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, pp. 1–7. Springer, 1995.

[AB98] Sarabjot S. Anand and Alex G. Büchner. *Decision Support Using Data Mining.* Financial Times Pitman Publishers, 1998.

[ABC08] Mohammad Assaad, Romuald Boné, and Hubert Cardot. "A New Boosting Algorithm for Improved Time-series Forecasting with Recurrent Neural Networks." *Inf. Fusion*, **9**(1):41–55, 2008.

[Abe03] Shigeo Abe. "Analysis of Multiclass Support Vector Machines." In *CIMCA '03: International Conference on Computational Intelligence for Modelling Control and Automation*, pp. 385–396, 2003.

[AC99] Aijun An and Nick Cercone. "Discretization of Continuous Attributes for Learning Classification Rules." In *PAKDD '99: Proceedings of the 3rd Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, pp. 509–514. Springer, 1999.

[Aha92] David W. Aha. "Tolerating Noisy, Irrelevant and Novel Attributes in Instance-based Learning Algorithms." *Int. J. Man-Mach. Stud.*, **36**(2):267–287, 1992.

[AKC00] Ion Androutsopoulos, John Koutsias, Konstantinos Chandrinos, and Constantine D. Spyropoulos. "An Experimental Comparison of Naive Bayesian and Keyword-based Anti-Spam Filtering with Personal E-mail Messages." In *SIGIR '00: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 160–167, 2000.

[AN09] Jorge M. Arevalillo and Hilario Navarro. "Using Random Forests to Uncover Bivariate Interactions in High Dimensional Small Data Sets." In *StReBio '09: Proceedings of the KDD-09 Workshop on Statistical and Relational Learning in Bioinformatics*, pp. 3–6. ACM, 2009.

[ANW07] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. "A Comparison of Machine Learning Techniques for Phishing Detection." In *eCrime '07: Proceedings of the Anti-phishing Working Groups 2nd Annual eCrime Researchers Summit*, pp. 60–69. ACM, 2007.

[Att99] H. Attias. "Independent factor analysis." *Neural Comput.*, **11**(4):803–851, 1999.

[BA97] Leonard A. Breslow and David W. Aha. "Simplifying Decision Trees: A Survey." *Knowl. Eng. Rev.*, **12**(1):1–40, 1997.

[Bac02] Adam Back. "HashCash – A Denial of Service Counter-Measure.", 2002. Available on-line: `http://www.hashcash.org/papers/hashcash.pdf` (visited 11/2009).

[BB05a] Michael W. Berry and Murray Browne. “Email Surveillance Using Non-negative Matrix Factorization.” *Comput. Math. Organ. Theory*, **11**(3):249–264, 2005.

[BB05b] Michael W. Berry and Murray Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools).* Society for Industrial and Applied Mathematics, 2nd edition, 2005.

[BBL07] Michael W. Berry, Murray Browne, Amy N. Langville, Paul V. Pauca, and Robert J. Plemmons. “Algorithms and Applications for Approximate Nonnegative Matrix Factorization.” *Computational Statistics & Data Analysis*, **52**(1):155–173, 2007.

[BC04] Jason R. Blevins and Moody T. Chu. “Updating the Centroid Decomposition with Applications in LSI.”, 2004. Unpublished manuscript, available on-line: `http://jblevins.org/research/centroid/` (visited 12/2009).

[BCF06] Andrej Bratko, Gordon V. Cormack, Bogdan Filipic, Thomas R. Lynam, and Blaz Zupan. “Spam Filtering Using Statistical Data Compression Models.” *J. Mach. Learn. Res.*, **6**:2673–2698, 2006.

[BDJ99] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. “Matrices, Vector Spaces, and Information Retrieval.” *SIAM Review*, **41**(2):335–362, 1999.

[BDL08] Gérard Biau, Luc Devroye, and Gábor Lugosi. “Consistency of Random Forests and Other Averaging Classifiers.” *J. Mach. Learn. Res.*, **9**:2015–2033, 2008.

[Bel61] Richard Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[Ber92] Michael W. Berry. “Large Scale Singular Value Computations.” *International Journal of Supercomputer Applications*, **6**:13–49, 1992.

[Ber00] Daniel J. Bernstein. “Internet Mail 2000.”, 2000. Available on-line: `http://cr.yp.to/im2000.html` (visited 12/2009).

[Ber02] Pavel Berkhin. “Survey of Clustering Data Mining Techniques.” Technical report, Accrue Software, 2002.

[BFS84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees.* Chapman and Hall, 1984.

[BG08] Christos Boutsidis and Efstratios Gallopoulos. “SVD based Initialization: A Head Start for Nonnegative Matrix Factorization.” *Pattern Recogn.*, **41**(4):1350–1362, 2008.

[BH08] Peter Bühlmann and Torsten Hothorn. “Boosting Algorithms: Regularization, Prediction and Model Fitting.” *Statistical Science*, **22**:477–505, 2008.

[Bis07] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 2007.

[BKK06] Emmanouil Benetos, Margarita Kotti, and Constantine Kotropoulos. “Applying Supervised Classifiers Based on Non-negative Matrix Factorization to Musical Instrument Classification.” *Multimedia and Expo, IEEE International Conference on*, **0**:2105 – 2108, 2006.

[BL97] Avrim L. Blum and Pat Langley. “Selection of Relevant Features and Examples in Machine Learning.” *Artificial Intelligence*, **97**:245–271, 1997.

[BL04] Michael J. A. Berry and Gordon S. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management.* Wiley Computer Publishing, 2004.

[Boy04] Jonathan de Boyne Pollard. “Fleshing out IM2000.”, 2004. Available on-line: `http://homepages.tesco.net/~J.deBoynePollard/Proposals/IM2000/` (visited 12/2009).

[Bra02] Matthew Brand. “Incremental Singular Value Decomposition of Uncertain Data with Missing Values.” In *ECCV ’02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pp. 707–720. Springer, 2002.

[Bra07] Max Bramer. *Principles of Data Mining.* Springer, 2007.

[Bra08] Roger B. Bradford. "An Empirical Study of Required Dimensionality for Large-Scale Latent Semantic Indexing Applications." In *CIKM '08: Proceeding of the 17th ACM Conference on Information and Knowledge Management*, pp. 153–162. ACM, 2008.

[Bre96] Leo Breiman. "Bagging Predictors." *Machine Learning*, **24**(2):123–140, 1996.

[Bre01] Leo Breiman. "Random Forests." *Machine Learning*, **45**(1):5–32, 2001.

[BS03] Alti J. Benediktsson and Johannes R. Sveinsson. "Multisource Remote Sensing Data Classification Based on Consensus and Pruning." *IEEE Transactions on Geoscience and Remote Sensing*, **41**(4):932–936, April 2003.

[BTG04] J. P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov. "Metagenes and molecular pattern discovery using matrix factorization." *Proc Natl Acad Sci U S A*, **101**(12):4164–4169, 2004.

[BTH01] Robert Burbidge, Matthew Trotter, Sean Holden, and Bernard Buxton. "Drug Design by Machine Learning: Support Vector Machines for Pharmaceutical Data Analysis." *Comput. Chem*, **26**:5–14, 2001.

[Bur98] Christopher J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition." *Data Min. Knowl. Discov.*, **2**(2):121–167, 1998.

[BWG08] Kanishka Bhaduri, Ran Wolff, Chris Giannella, and Hillol Kargupta. "Distributed Decision-Tree Induction in Peer-to-Peer Systems." *Stat. Anal. Data Min.*, **1**(2):85–103, 2008.

[BZF06] Robert F. Battisti, Yanqiang Zhong, Lanyan Fang, Seth Gibbs, Jie Shen, Janos Nadas, Guisheng Zhang, and Duxin Sun. "Modifying the Sugar Moieties of Daunorubicin Overcomes P-gp-Mediated Multidrug Resistance." *J. Mol. Pharm*, **220**(4598):671–680, 2006.

[CB91] Peter Clark and Robin Boswell. "Rule Induction with CN2: Some Recent Improvements." In *EWSL-91: Proceedings of the European Working Session on Learning on Machine Learning*, pp. 151–163. Springer, 1991.

[CC01] Trevor F. Cox and Michael A. A. Cox. *Multidimensional Scaling.* Chapman & Hall, 2001.

[CCK00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. "CRISP-DM 1.0 Step-by-Step Data Mining Guide." Technical report, The CRISP-DM consortium, 2000.

[CCU06] Madhusudhanan Chandrasekaran, Ramkumar Chinchani, and Shambhu Upadhyaya. "PHONEY: Mimicking User Response to Detect Phishing Attacks." In *WOWMOM' 06: Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, pp. 668–672. IEEE Computer Society, 2006.

[CF01] Moody T. Chu and Robert E. Funderlic. "The Centroid Decomposition: Relationships between Discrete Variational Decompositions and SVDs." *SIAM J. Matrix Anal. Appl.*, **23**(4):1025–1044, 2001.

[CFV07] Pablo Castells, Miriam Fernandez, and David Vallet. "An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval." *IEEE Transactions on Knowledge and Data Engineering*, **19**(2):261–272, 2007.

[CGO06] Patricia de Cerqueira Lima, Alexander Golbraikh, Scott Oloff, Yunde Xiao, and Alexander Tropsha. "Combinatorial QSAR Modeling of P-Glycoprotein Substrates." *J. Chem. Inf. Model.*, **46**(3):1245–1254, 2006.

[CH67] Thomas M. Cover and P. E. Hart. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, **13**(1):21–27, 1967.

[CH95] Thomas M. Cover and P. E. Hart. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, **8**(6):373–389, 1995.

[Cha92] Tony Chan. "An Improved Algorithm for Computing the Singular Value Decomposition." *Trans. Math. Soft.*, **8**:72 – 83, 1992.

[Che97] Steve Cherry. "Some Comments on Singular Value Decomposition Analysis." *Journal of Climate*, **10**(7):1759–1761, 1997.

[Che09] ChemDiv. "ChemDiv: Chemical Shop.", 2009. Available on-line: `http://chemdiv.emolecules.com` (visited 12/2009).

[CHN04] M. Catral, Lixing Han, Michael Neumann, and R. J. Plemmons. "On reduced rank nonnegative matrix factorization for symmetric nonnegative matrices." *Linear Algebra and its Applications*, **393**:107 – 126, 2004. Special Issue on Positivity in Linear Algebra.

[CHS97] Peter Cabena, Pablo Hadjnian, Rolf Stadler, Jaap Verhees, and Alessandro Zanasi. *Discovering Data Mining: From Concept to Implementation.* Prentice Hall, 1997.

[CKY08] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. "An Empirical Evaluation of Supervised Learning in High Dimensions." In Andrew McCallum and Sam Roweis, editors, *ICML' 2008: Proceedings of the 25th International Conference on Machine Learning*, pp. 96–103. Omnipress, 2008.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (visited 12/2009).

[CL05] Gordon V. Cormack and Thomas R. Lynam. "TREC 2005 Spam Public Corpora.", 2005. Available on-line: `http://plg.uwaterloo.ca/cgi-bin/cgiwrap/gvcormac/foo` (visited 12/2009).

[Coh95] William W. Cohen. "Fast Effective Rule Induction." In *ICML' 95: Proceedings of the 12th International Conference on Machine Learning*, pp. 115–123. Morgan Kaufmann, 1995.

[Coh09] William W. Cohen. "The Enron Email Dataset.", 2009. Available on-line: `http://www.cs.cmu.edu/~enron` (visited 12/2009).

[Cor07] Gordon V. Cormack. "Email Spam Filtering: A Systematic Review." *Foundations and Trends in Information Retrieval*, **1**(4):335–455, 2007.

[CPS07] Krzysztof J. Cios, Witold Pedrycz, Roman W. Swiniarski, and Lukasz A. Kurgan. *Data Mining: A Knowledge Discovery Approach.* Springer, 2007.

[CS93] Scott Cost and Steven Salzberg. "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features." *Mach. Learn.*, **10**(1):57–78, 1993.

[CS00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[CS02] Koby Crammer and Yoram Singer. "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines." *J. Mach. Learn. Res.*, **2**:265–292, 2002.

[Cup81] J. J. M. Cuppen. "A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem." *Numer. Math.*, **36**:177–195, 1981.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks." *Machine Learning*, **20**(3):273–297, 1995.

[CXM05] Zhan Chuan, Lu Xianliang, Hou Mengshu, and Zhou Xu. "A LVQ-based Neural Network Anti-spam Email Approach." *SIGOPS Oper. Syst. Rev.*, **39**(1):34–39, 2005.

[DA06] Ramon Diaz-Uriarte and Sara Alvarez de Andres. "Gene Selection and Classification of Microarray Data Using Random Forest." *BMC Bioinformatics*, **7**(1):3, 2006.

[Das01] Sanmay Das. "Filters, Wrappers and a Boosting-based Hybrid for Feature Selection." In *ICML '01: Proceedings of the 18th International Conference on Machine Learning*, pp. 74–81. Morgan Kaufmann Publishers Inc., 2001.

[DCC05] Sarah Jane Delany, Pádraig Cunningham, and Lorcan Coyle. "An Assessment of Case-Based Reasoning for Spam Filtering." *Artif. Intell. Rev.*, **24**(3-4):359–378, 2005.

[DCD05] Sarah Jane Delany, Padraig Cunningham, Dónal Doyle, and Anton Zamolotskikh. "Generating Estimates of Classification Confidence for a Case-Based Spam Filter." In *ICCBR 05: Proceedings of the 6th International Conference on Case-Based Reasoning*, pp. 177–190, 2005.

[DDF90] Scott C. Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard A. Harshman. "Indexing by Latent Semantic Analysis." *Journal of the American Society for Information Science*, **41**:391–407, 1990.

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[Det04] Marcel Dettling. "BagBoosting for Tumor Classification with Gene Expression Data." *Bioinformatics*, **20**(18):3583+, 2004.

[DF95] Karsten M. Decker and Sergio Focardi. "Technology Overview: a Report on Data Mining." Technical report, Swiss Federal Institute of Technology (ETH Zurich) Technical Report CSCS TR-95-02, 1995.

[DGG07] Daniel Delling, Marco Gaertler, Robert Görke, Zoran Nikoloski, and Dorothea Wagner. "How to Evaluate Clustering Techniques." Technical report, University of Karlsruhe, Facultiy of Computer Science, 2007.

[DGN03] Cynthia Dwork, Andrew Goldberg, and Moni Naor. "On Memory-Bound Functions for Fighting Spam." In *In Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003)*, pp. 426–444. Springer, 2003.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition).* Wiley-Interscience, 2001.

[Die98] Thomas G. Dietterich. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Neural Computation*, **10**:1895–1923, 1998.

[Die00] Thomas G. Dietterich. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization." *J. Mach. Learn. Res.*, **40**(2):139–157, 2000.

[Die02] Thomas G. Dietterich. "Ensemble Learning." In *The Handbook of Brain Theory and Neural Networks*, pp. 405–406. MIT Press, Cambridge, MA, 2nd edition, 2002.

[DIP06] Elodie Dubus, Ismail Ijjaali, Francois Petitet, and Andre Michel. "In Silico Classification of hERG Channel Blockers: a Knowledge-Based Strategy." *ChemMedChem*, **1**(7):662, 2006.

[DJT08] Michael A. Demel, Andreas G.K. Janecek, Khac-Minh Thai, Gerhard F. Ecker, and Wilfried N. Gansterer. "Predictive QSAR Models for Polyspecific Drug Targets: The Importance of Feature Selection." *Current Computer - Aided Drug Design*, **4**(2):91–110, 2008.

[DL03] Manoranjan Dash and Huan Liu. "Consistency-based Search in Feature Selection." *Artif. Intell.*, **151**(1-2):155–176, 2003.

[DLM00] Manoranjan Dash, Huan Liu, and Hiroshi Motoda. "Consistency-based Feature Selection." In *PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pp. 98–109. Springer, 2000.

[DLP06] Chris Ding, Tao Li, Wei Peng, and Haesun Park. "Orthogonal nonnegative matrix t-factorizations for clustering." In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 126–135, 2006.

[DM01] Inderjit S. Dhillon and Dharmendra S. Modha. "Concept Decompositions for Large Sparse Text Data Using Clustering." *J. Mach. Learn. Res.*, **42**(1):143–175, 2001.

[DMP08] James W. Demmel, Osni A. Marques, Beresford N. Parlett, and Christof Vömel. "Performance and Accuracy of LAPACK's Symmetric Tridiagonal Eigensolvers." *SIAM J. Sci. Comput.*, **30**(3):1508–1526, 2008.

[Dom99] Pedro Domingos. "MetaCost: A General Method for Making Classifiers Cost-Sensitive." In *SIGKDD '99: Proceedings of the 5th International ACM Conference on Knowledge Discovery and Data Mining*, pp. 155–164. ACM Press, 1999.

[DS05] Inderjit S. Dhillon and Suvrit Sra. "Generalized Nonnegative Matrix Approximations with Bregman Divergences." *Advances in Neural Information Processing Systems*, **18**:283–290, 2005.

[Dum91] Susan T. Dumais. "Improving the Retrieval of Information from External Sources." *Behavior Research Methods, Instruments, & Computers*, **23**(2):229–236, 1991.

[Dun02] Margaret H. Dunham. *Data Mining: Introductory and Advanced Topics.* Prentice Hall, 2002.

[DW08] Karthik Devarajan and Guoli Wang. "Parallel Implementation of Nonnegative Matrix Factorization (NMF) Algorithms using High-Performance Computing (HPC) Cluster." In *Proceedings of the 39th Symposium on the Interface: Computing Science and Statistics. Theme: Systems Biology*, pp. 23–26, 2008.

[Dy07] Jennifer G. Dy. "Unsupervised Feature Selection." In *Computational Methods of Feature Selection*, pp. 19–39. in Computational Methods of Feature Selection edited by Huan Liu and Hiroshi Motoda, Chapman and Hall/CRC Press, 2007.

[DZ04] Saso Dzeroski and Bernard Zenko. "Is Combining Classifiers with Stacking Better than Selecting the Best One?" *Mach. Learn.*, **54**(3):255–273, 2004.

[DZW09] Chao Dong, Huijie Zhao, and Wei Wang. "Parallel Nonnegative Matrix Factorization Algorithm on the Distributed Memory Platform." *International Journal of Parallel Programming*, 2009.

[EBS06] Sean Ekins, Konstantin V. Balakin, Nikolay Savchuk, and Yan Ivanenkov. "Insights for Human Ether-a-go-go-related Gene Potassium Channel Inhibition using Recursive Partitioning and Kohonen and Sammon Mapping Techniques." *J. Med. Chem.*, **49**(17):5059–71, 2006.

[Edu09] Edusoft-LC. "Molconn-Z." Available on-line: `http://www.edusoft-lc.com/molconn` (visited 12/2009), 2009.

[ES05] Roberto Esposito and Lorenza Saitta. "Experimental Comparison Between Bagging and Monte Carlo Ensemble Classification." In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pp. 209–216. ACM, 2005.

[EY36] Carl Eckart and Gale Young. "The Approximation of One Matrix by Another of Lower Rank." *Psychometrika*, **1**(3):211–218, 1936.

[FC04] George Forman and Ira Cohen. "Learning from Little: Comparison of Classifiers Given Little Training." In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 161–172. Springer, 2004.

[FC08] Kai-Yao Chang Fi-John Chang and Li-Chiu Chang. "Counterpropagation Fuzzy-neural Network for City Flood Control System." *Journal of Hydrology*, **358**(1-20):24–34, 2008.

[FGW01] Usama Fayyad, Georges G. Grinstein, and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery.* Morgan Kaufmann Publishers Inc., 2001.

[FHR02] César Ferri, José Hernández-Orallo, and M. José Ramírez-Quintana. "From Ensemble Methods to Comprehensible Models." In *DS '02: Proceedings of the 5th International Conference on Discovery Science*, pp. 165–177. Springer, 2002.

[FID07] Florentino Fdez-Riverola, Eva L. Iglesias, Fernando Diaz, José R. Mendez, and Juan M. Corchado. "SpamHunting: An Instance-Rased reasoning System for Spam Labelling and Filtering." *Decis. Support Syst.*, **43**(3):722–736, 2007.

[FL95] Christos Faloutsos and King-Ip Lin. "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets." In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 163–174. ACM, 1995.

[Fle04] Francois Fleuret. "Fast Binary Feature Selection with Conditional Mutual Information." *J. Mach. Learn. Res.*, **5**:1531–1555, 2004.

[FPS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery: An Overview." In *Advances in Knowledge Discovery and Data Mining*, pp. 1–34. Menlo Park, AAAI Press, 1996.

[FS97] Yoav Freund and Robert E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Science*, **55**(1):119–139, 1997.

[FS06] Ronen Feldman and James Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, 2006.

[FST07] Ian Fette, Norman Sadeh, and Anthony Tomasic. "Learning to Detect Phishing Emails." In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pp. 649–656. ACM, 2007.

[Fue99] Johannes Fuernkranz. "Separate-and-Conquer Rule Learning." *Artif. Intell. Rev.*, **13**(1):3–54, 1999.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition.* Academic Press, 1990.

[FW94] Johannes Fürnkranz and Gerhard Widmer. "Incremental Reduced Error Pruning." In *ICML '94: Proceedings of the 11th International Conference of Machine Learning*, pp. 70–77, 1994.

[FYS09] Weiwei Fang, Bingru Yang, Dingli Song, and Zhigang Tang. "A New Scheme on Privacy-Preserving Distributed Decision-Tree Mining." In *ETCS '09: Proceedings of the 2009 First International Workshop on Education Technology and Computer Science*, pp. 517–520. IEEE Computer Society, 2009.

[Gar07] Al Gardner. "defNULLspam." `http://www.defnullspam.com/`, 2007.

[GCL09] Yongming Guo, Dehua Chen, and Jiajin Le. "An Extended Vector Space Model for XML Information Retrieval." In *WKDD '09: Proceedings of the 2009 Second International Workshop on Knowledge Discovery and Data Mining*, pp. 797–800. IEEE Computer Society, 2009.

[GDB08] Vincent Garcia, Eric Debreuve, and Michel Barlaud. "Fast k-Nearest-Neighbor Search Using GPU." In *Proceedings of the CVPR Workshop on Computer Vision on GPU*, 2008.

[GDD01] Arthur Gretton, Manuel Davy, Arnaud Doucet, and Peter J. W. Rayner. "Nonstationary Signal Classification Using Support Vector Machines." In *IEEE Workshop on Statistical Signal Processing Proceedings*, pp. 305–308, 2001.

[GE03] Isabelle Guyon and Andre Elisseeff. "An Introduction to Variable and Feature Selection." *J. Mach. Learn. Res.*, **3**:1157–1182, 2003.

[Gee03] Kevin R. Gee. "Using Latent Semantic Indexing to Filter Spam." In *ACM Symposium on Applied Computing, Data Mining Track*, pp. 460–464, 2003.

[GG91] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1991.

[GGM08] Michael S. Gashler, Christophe Giraud-Carrier, and Tony Martinez. "Decision Tree Ensemble: Small Heterogeneous is Better than Large Homogeneous." In *ICMLA '08: Proceedings of the 7th International Conference on Machine Learning and Applications*, pp. 900–905, 2008.

[GGN06] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing).* Springer, 2006.

[GHI05] Wilfried N. Gansterer, Helmut Hlavacs, Michael Ilger, Peter Lechner, and Jürgen Strauß. "Token Buckets for Outgoing Spam Prevention." In *CNIS '05: Proceedings of the IASTED International Conference on Communication, Network, and Information Security.* ACTA Press, 2005.

[GIL05] Wilfried N. Gansterer, Michael Ilger, Peter Lechner, Robert Neumayer, and Jürgen Strauß. "Anti-Spam Methods - State of the Art." Technical Report FA384018-1, Institute of Distributed and Multimedia Systems, Faculty of Computer Science, University of Vienna, 2005.

[GJL07] Wilfried N. Gansterer, Andreas G.K. Janecek, and Peter Lechner. "A Reliable Component-based Architecture for E-Mail Filtering." In *ARES '07: Proceedings of the 2nd International Conference on Availability, Reliability and Security*, pp. 43–50. IEEE Computer Society, 2007.

[GJN08] Wilfried N. Gansterer, Andreas G.K. Janecek, and Robert Neumayer. "Spam Filtering Based on Latent Semantic Indexing." In Michael W. Barry and Malu Castellanos, editors, *Survey of Text Mining II: Clustering, Classification, and Retrieval*, pp. 165–183. Springer, 2008.

[GK86] Paul Geladi and Bruce R. Kowalski. "Partial Least-squares Regression: A Tutorial." *Analytica Chimica Acta*, **185**:1–17, 1986.

[GKK03] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. *Introduction to Parallel Computing (2nd Edition).* Addison Wesley, 2003.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1989.

[Gom04] Jonatan Gomez. "Evolution of Fuzzy Rule Based Classifiers." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2004*, pp. 1150–1161, 2004.

[Goo09] Google. "Google Safe Browsing for Firefox.", 2009. Available on-line: `http://www.google.com/tools/firefox/safebrowsing` (visited 12/2009).

[Gor83] Richard L. Gorsuch. *Factor Analysis.* Lawrence Erlbaum, 2nd edition, 1983.

[GP09] Wilfried N. Gansterer and David Pölz. "E-Mail Classification for Phishing Defense." In *ECIR '09: Proceedings of the 31th European Conference on Information Retrieval Research*, Lecture Notes in Computer Science (5478), pp. 449–460. Springer, 2009.

[Gro09] Chemical Computing Group. "Molecular Operating Environment (MOE)." Available on-line: `http://www.chemcomp.com/software.htm` (visited 12/2009), 2009.

[GSV02] David Guillamet, Bernt Schiele, and Jordi Vitria. "Analyzing Non-Negative Matrix Factorization for Image Classification." In *ICPR '02: In Proceedings of the 16th International Conference on Pattern Recognition*, pp. 116–119, 2002.

[Gup09] Satya P. Gupta. *QSAR and Molecular Modeling.* Springer, 2009.

[Guy08] Isabelle Guyon. "Practical Feature Selection: from Correlation to Causality." In *Mining Massive Data Sets for Security: Advances in Data Mining, Search, Social Networks and Text Mining, and their Applications to Security*, pp. 27–43. IOS Press, 2008.

[GV96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences).* The Johns Hopkins University Press, October 1996.

[GVS03] David Guillamet, Jordi Vitria, and Bernt Schiele. "Introducing a weighted non-negative matrix factorization for image classification." *Pattern Recognition Letters*, **24**(14):2447 – 2454, 2003.

[GWB02] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. "Gene Selection for Cancer Classification using Support Vector Machines." *Machine Learning*, **46**(1):389–422, 2002.

[Hal00] Mark A. Hall. "Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning." In *ICML '00: Proceedings of the 17th International Conference on Machine Learning*, pp. 359–366. Morgan Kaufmann Publishers Inc., 2000.

[Har03] E. Harris. "The Next Step in the Spam Control War: Greylisting." Technical report, PureMagic Software, 2003. `http://projects.puremagic.com/greylisting/whitepaper.html`.

[HDB05] Torsten Hothorn, Marcel Dettling, and Peter Bühlmann. "Ensemble Methods of Computational Inference." In *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pp. 293–312. Springer, 2005.

[Hec87] Robert Hecht-Nielsen. "Counterpropagation Networks." *Applied Optics*, pp. 4979–4984, 1987.

[HFS01] Richard D. Hull, Eugene M. Fluder, Suresh B. Singh, Robert B. Nachbar, Robert P. Sheridan, and Simon K. Kearsley. "Latent semantic structure indexing (LaSSI) for defining chemical similarity." *J Med Chem*, **44**(8):1177–1184, 2001.

[HHW05] Ramin Homayouni, Kevin Heinrich, Lai Wei, and Michael W. Berry. "Gene Clustering by Latent Semantic Indexing of MEDLINE Abstracts." *Bioinformatics*, **21**(1):104–115, 2005.

[Hid05] José Maria Gomez Hidalgo. "Machine Learning for Spam Detection References.", 2005. Available on-line: `http://www.esi.uem.es/~jmgomez/spam/MLSpamBibliography.htm` (visited 12/2009).

[Hir97] Sugihara Hiroshi. "What is Occam's Razor?", 1997. Available on-line: `http://math.ucr.edu/home/baez/physics/General/occam.html` (visited 12/2009).

[HK06] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques, 2nd ed.* Morgan Kaufmann Publishers, 2006.

[HKK01] Eui-Hong Han, George Karypis, and Vipin Kumar. "Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification." In *PAKDD '01: Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 53–65. Springer, 2001.

[HKK06] Te-Ming Huang, Vojislav Kecman, and Ivica Kopriva. *Kernel Based Algorithms for Mining Huge Data Sets: Supervised, Semi-supervised, and Unsupervised Learning (Studies in Computational Intelligence).* Springer, 2006.

[HKO01] Aapo Hyvarinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis.* J. Wiley, 2001.

[HL95] Richard J. Hanson and Charles L. Lawson. *Solving Least Squares Problems.* SIAM, 1995.

[HMM07] Jianping Huang, Guangli Ma, Ishtiaq Muhammad, and Yiyu Cheng. "Identifying P-Glycoprotein Substrates Using a Support Vector Machine Optimized by a Particle Swarm." *J. Chem. Inf. Model.*, **47**(4):1638–1647, 2007.

[HMO06] Tom Howley, Michael G. Madden, Marie-Louise O'Connell, and Alan G. Ryder. "The Effect of Principal Component Analysis on Machine Learning Accuracy with High-dimensional Spectral Data." *Knowledge Based Systems*, **19**(5):363–370, 2006.

[Hol93] Robert C. Holte. "Very Simple Classification Rules Perform Well on Most Commonly Used Datasets." *J. Mach. Learn. Res.*, **11**(1):63–90, 1993.

[Hoy04] Patrik O. Hoyer. "Non-negative Matrix Factorization with Sparseness Constraints." *Journal of Machine Learning Research*, **5**:1457–1469, 2004.

[HP07] Xiaohua Hu and Yi Pan. *Knowledge Discovery in Bioinformatics: Techniques, Methods, and Applications (Wiley Series in Bioinformatics).* John Wiley & Sons, Inc., 2007.

[HSL02] William H. Hsu, Cecil P. Schmidt, and James A. Louis. "Genetic Algorithm Wrappers For Feature Subset Selection In Supervised Inductive Learning." In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, p. 680. Morgan Kaufmann Publishers Inc., 2002.

[HSN01] Richard D. Hull, Suresh B. Singh, Robert B. Nachbar, Robert P. Sheridan, Simon K. Kearsley, and Eugene M. Fluder. "Chemical Similarity Searches using Latent Semantic Structure Indexing (LaSSI) and Comparison to TOPOSIM." *J. Med. Chem.*, **44**(8):1185–1191, 2001.

[HSR08] Hans-Dieter Höltje, Wolfgang Sippl, Didier Rognan, and Gerd Folkers. *Molecular Modeling: Basic Principles and Applications, 3rd Edition.* John Wiley & Sons, Inc., 2008.

[HTF02] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2002.

[Ioa03] John Ioannidis. "Fighting Spam by Encapsulating Policy in Email addresses." In *Proceedings of the 10th Annual Network and Distributed Systems Security Conference (NDSS 2003)*, pp. 8–8, 2003.

[ISG06] Michael Ilger, Jürgen Strauß, Wilfried N. Gansterer, and Christian Proschinger. "The Economy of Spam." Technical Report FA384018-6, Institute of Distributed and Multimedia Systems, Faculty of Computer Science, University of Vienna, 2006.

[Iva07] Ovidiu Ivanciuc. "Applications of Support Vector Machines in Chemistry." *Rev. Comput. Chem.*, **23**:291–400, 2007.

[JC99] Jacek Jarmulak and Susan Craw. "Genetic Algorithms for Feature Selection and Weighting." In *IJCAI '99: Proceedings of the 1999 Workshop on Automating the Construction of Case Based Reasoners*, pp. 28–33, 1999.

[JCW07] Liangxiao Jiang, Zhihua Cai, Dianhong Wang, and Siwei Jiang. "Survey of Improving K-Nearest-Neighbor for Classification." *Fuzzy Systems and Knowledge Discovery, Fourth International Conference on*, **1**:679–683, 2007.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[Jen77] Robert I. Jennrich. "Stepwise Discriminant Analysis." In *Methods for Digital Computers*, volume 3, pp. 76–96. Wiley, New York, 1977.

[JF87] William P. Jones and George W. Furnas. "Pictures of Relevance: A Geometric Analysis of Similarity Measures." *Journal of the American Society for Information Science*, **38**(6):420–442, 1987.

[JG09] Andreas G.K. Janecek and Wilfried N. Gansterer. "E-Mail Classification based on Non-Negative Matrix Factorization." In *Text Mining 2009*, 2009.

[JGD08] Andreas G.K. Janecek, Wilfried N. Gansterer, Michael Demel, and Gerhard F. Ecker. "On the Relationship Between Feature Selection and Classification Accuracy." *JMLR: Workshop and Conference Proceedings*, **4**:90–105, 2008.

[JGK08] Andreas G.K. Janecek, Wilfried N. Gansterer, and K. Ashwin Kumar. "Multi-Level Reputation-Based Greylisting." In *ARES '08: Proceedings of the 3rd International Conference on Availability, Reliability and Security*, pp. 10–17. IEEE Computer Society, 2008.

[Jia04] Wenxin Jiang. "Boosting with Noisy Data: Some Views from Statistical Theory." *Neural Comput.*, **16**(4):789–810, 2004.

[Joa98] Thorsten Joachims. "Text Categorization with Support Vector Machines: Learning with Many Relevant Features." In *ECML'98, Proceedings of the 10th European Conference on Machine Learning*, pp. 137–142. Springer, 1998.

[Jol02] Ian T. Jolliffe. *Principal Component Analysis.* Springer, 2nd edition, 2002.

[Jon72] Karen Sparck Jones. "A Statistical Interpretation of Term Specificity and its Application to Retrieval." *Journal of Documentation*, **28**(1):11–20, 1972.

[JW07] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis (6th Edition).* Prentice Hall, 2007.

[Kan07] Khushboo Kanjani. "Parallel Non Negative Matrix Factorization for Document Clustering.", 2007. Available on-line: `http://parasol.tamu.edu/people/khush/DocumentClustering.pdf` (visited 12/2009).

[Kas80] Gordon V. Kass. "An Exploratory Technique for Investigating Large Quantities of Categorical Data." *Journal of Applied Statistics*, **29**(2):119–127, 1980.

[KC00] Hillol Kargupta and Philip Chan. *Advances in Distributed and Parallel Knowledge Discovery.* AAAI Press, 2000.

[Kes03] György M. Keserü. "Prediction of hERG Potassium Channel Affinity by Traditional and Hologram qSAR Methods." *Bioorg. Med. Chem. Lett.*, **13**(16):2773–5, Aug 18 2003.

[KGV83] Scott Kirkpatrick, Daniel C. Gelatt, and Mario P. Vecchi. "Optimization by Simulated Annealing." *Science*, **220**:661–680, 1983.

[KHY08] Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar. *Next Generation of Data Mining.* Chapman & Hall, 2008.

[KJ97] Ron Kohavi and George H. John. "Wrappers for Feature Subset Selection." *Artificial Intelligence*, **97**(1-2):273–324, 1997.

[KJS04] Hillol Kargupta, Anupam Joshi, Krishnamoorthy Sivakumar, and Yelena Yesha. *Data Mining: Next Generation Challenges and Future Directions.* AAAI/MIT Press, 2004.

[KL93] Shigeru Katagiri and Chin-Hui Lee. "A New Hybrid Algorithm for Speech Recognition Based on HMM Segmentation and Learning Vector Quantization." *Speech and Audio Processing, IEEE Transactions on*, **1**(4):421–430, 1993.

[KLL03] Oleg Kolesnikov, Wenke Lee, and Richard Lipton. "Filtering spam using search engines." Technical report, Georgia Institute of Technology, 2003.

[Koh95] Ron Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection." In *IJCAI '95: Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pp. 1137–1145, 1995.

[Koh00] Teuvo Kohonen. *Self-Organizing Maps.* Springer, 3rd edition, 2000.

[KOS09] Andreas Köhler, Matthias Ohrnberger, and Frank Scherbaum. "Unsupervised Feature Selection and General Pattern Discovery using Self-organizing Maps for Gaining Insights into the Nature of Seismic Wavefields." *Comput. Geosci.*, **35**(9):1757–1767, 2009.

[Kot07] Sotiris B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques." *Informatica*, **31**(3):249–268, 2007.

[KP04] Sotiris B. Kotsiantis and Panayiotis E. Pintelas. "Increasing the Classification Accuracy of Simple Bayesian Classifier." *Lecture Notes in Artificial Intelligence*, **31**(3192):198–207, 2004.

[KP08] Hyunsoo Kim and Haesun Park. "Nonnegative Matrix Factorization Based on Alternating Nonnegativity Constrained Least Squares and Active Set Method." *SIAM J. Matrix Anal. Appl.*, **30**(2):713–730, 2008.

[KR92] Kenji Kira and Larry A. Rendell. "A Practical Approach to Feature Selection." In *Proceedings of the 9th International Workshop on Machine Learning.* Morgan Kaufmann Publishers Inc., 1992.

[KR05] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley, 2005.

[Kra98] Richard Kramer. *Chemometric Techniques for Quantitative Analysis.* Marcel-Dekker, 1998.

[KS93] Ben J.A . Kroese and Patrick P. van der Smagt. *An Introduction to Neural Networks.* The University of Amsterdam, 1993.

[KS96] Daphne Koller and Mehran Sahami. "Toward Optimal Feature Selection." Technical report, Stanford InfoLab, 1996.

[Lab99] Paul Labute. "Binary QSAR: A New Method for the Determination of Quantitative Structure Activity Relationships." In *Pacific Symposium on Biocomputing*, pp. 444–455, 1999.

[Lai07] Chih-Chin Lai. "An Empirical Study of Three Machine Learning Methods for Spam Filtering." *Know.-Based Syst.*, **20**(3):249–254, 2007.

[Lan05] Amy N. Langville. "The Linear Algebra Behind Search Engines." *Journal of Online Mathematics and its Applications (JOMA), Online Module*, 2005.

[Lar07] Daniel T. Larose. *Data Mining Methods and Models.* John Wiley & Sons, Inc., 2007.

[LBG80] Y. Linde, A. Buzo, and Robert M. Gray. "An Algorithm for Vector Quantizer Design." *IEEE Transactions on Communications*, pp. 702–710, 1980.

[LCL07] Xin Li, William K. W. Cheung, Jiming Liu, and Zhili Wu. "A Novel Orthogonal NMF-based Belief Compression for POMDPs." In *ICML '07: Proceedings of the 24th International Conference on Machine Learning*, pp. 537–544. ACM, 2007.

[LDH06] Wenyin Liu, Xiaotie Deng, Guanglin Huang, and Anthony Y. Fu. "An Antiphishing Strategy Based on Visual Similarity Assessment." *IEEE Internet Computing*, **10**(2):58–65, 2006.

[LDL97] Michael L. Littman, Susan T. Dumais, and Thomas K. Landauer. "Automatic Cross-linguistic Information Retrieval Using Latent Semantic Indexing." In *SIGIR '97: Proceedings of the 19th ACM Conference on Research and Development in Information Retrieval*, pp. 16–23, 1997.

[Lea01] Andrew Leach. *Molecular Modelling: Principles and Applications (2nd Edition).* Prentice Hall, 2001.

[Lev05] John R. Levine. "Experiences with Greylisting." In *CEAS '05: Proceedings of the 2nd Conference on Email and Anti-spam*, 2005.

[Lin07] Chih-Jen Lin. "Projected Gradient Methods for Nonnegative Matrix Factorization." *Neural Comput.*, **19**(10):2756–2779, 2007.

[Liu07] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications).* Springer, 2007.

[LL90] Thomas K. Landauer and Michael L. Littman. "Fully Automatic Cross-Language Document Retrieval Using Latent Semantic Indexing." In *Proceedings of the 6th Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research*, pp. 31–38, 1990.

[LLD04] Thomas K. Landauer, Darrell Laham, and Marcia Derr. "From Paragraph to Graph: Latent Semantic Analysis for Information Visualization." *Proceedings of the National Academy of Sciences of the United States of America*, **101**:5214–5219, 2004.

[LM98] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, 1998.

[LMA06] Amy N. Langville, Carl D. Meyer, and Russel Albright. "Initializations for the Nonnegative Matrix Factorization." In *SIGKDD '06: Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining*, 2006.

[LS96] Huan Liu and Rudy Setiono. “A Probabilistic Approach to Feature Aelection – A Filter Solution.” In *ICML ’96: Proceedings of the 13th International Conference on Machine Learning*, pp. 319–327. Morgan Kaufmann, 1996.

[LS99] Daniel D. Lee and H. Sebastian Seung. “Learning Parts of Objects by Non-negative Matrix Factorization.” *Nature*, **401**(6755):788–791, October 1999.

[LS01] Daniel D. Lee and H. Sebastian Seung. “Algorithms for Non-negative Matrix Factorization.” *Advances in Neural Information Processing Systems*, **13**:556–562, 2001.

[Ltd09] Strand Life Sciences Pvt. Ltd. “Qsar World.” Available on-line: `http://www.qsarworld.com/insilico-chemistry-chemical-descriptors.php` (visited 12/2009), 2009.

[LWL03] Hua Li, Xi-Zhao Wang, and Yong Li. “Using mutual information for selecting continuous-valued attribute in decision tree learning.” In *ICMLC’ 03: Proceedings of the 2003 International Conference on Machine Learning and Cybernetics*, pp. 1496–1499, 2003.

[LY05a] Fan Li and Yiming Yang. “Using Recursive Classification to Discover Predictive Features.” In *SAC ’05: Proceedings of the 2005 ACM Symposium on Applied Computing*, pp. 1054–1058. ACM, 2005.

[LY05b] Huan Liu and Lei Yu. “Toward Integrating Feature Selection Algorithms for Classification and Clustering.” *IEEE Trans. on Knowl. and Data Eng.*, **17**(4):491–502, 2005.

[LYU07a] H. Li, C.W. Yap, C.Y. Ung, Y. Xue, Z.R. Li, L.Y. Han, H.H. Lin, and Yu Zong Chen. “Machine Learning Approaches for Predicting Compounds that Interact with Therapeutic and ADMET Related Proteins.” *Journal of Pharmaceutical Sciences*, **96**(11):2838–2860, 2007.

[LYU07b] H. Li, C.W. Yap, C.Y. Ung, Y. Xue, Z.R. Li, L.Y. Han, H.H. Lin, and Yu Zong Chen. “MODEL – Molecular Descriptor Lab: A Web-based Server for Computing Structural and Physicochemical Features of Compounds.” *Journal of Biotechnology and Bioengineering*, **2**(97):389–396, 2007.

[Mat09a] The Mathworks. “Matlab Parallel Computing.” Available on-line, 2009. `http://www.mathworks.com/products/parallel-computing` (visited 12/2009).

[Mat09b] The Mathworks. “Matlab Release Notes.” Available on-line, 2009. `http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/rn.pdf` (visited 12/2009).

[MBN02] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. “Feature Selection Algorithms: A Survey and Experimental Evaluation.” In *ICDM ’02: Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 306–313. IEEE Computer Society, 2002.

[McA09] McAfee. “SiteAdvisor-Software.”, 2009. Available on-line: `http://spamassassin.apache.org/` (visited 12/2009).

[McG07] Colin McGregor. “Controlling Spam with SpamAssassin.” *Linux Journal*, **2007**(153):9, 2007.

[McL04] G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition.* Wiley Interscience, 2004.

[MCN08] E. Mejia-Roa, P. Carmona-Saez, R. Nogales, C. Vicente, M. Vazquez, X. Y. Yang, C. Garcia, F. Tirado, and A. Pascual-Montano. “bioNMF: A Web-based Tool for Nonnegative Matrix Factorization in Biology.” *Nucleic Acids Research.*, **36**:W523–W528, 2008.

[Mes09] MessageLabs. “MessageLabs Intelligence.”, 2009. Available on-line: `http://www.messagelabs.com/intelligence.aspx` (visited 12/2009).

[Mit98] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Education (ISE Editions), 1998.

[ML07] Zdravko Markov and Daniel T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage.* Wiley-Interscience, 2007.

[Mla06] Dunja Mladenic. "Feature Selection for Dimensionality Reduction." *Subspace, Latent Structure and Feature Selection*, pp. 84–102, 2006.

[MPI09] MPI Forum MPI. "Message Passing Interface Forum." Available on-line, 2009. `http://www.mpi-forum.org` (visited 12/2009).

[MR03] Ron Meir and Gunnar Rätsch. "An Introduction to Boosting and Leveraging." *Advanced lectures on machine learning*, pp. 118–183, 2003.

[Mur98] Sreerama K. Murthy. "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey." *Data Min. Knowl. Discov.*, **2**(4):345–389, 1998.

[NIS07] NIST. "TREC Tracks.", 2007. Available on-line: `http://trec.nist.gov/data/spam.html` (visited 12/2009).

[NIS09] NIST. "TREC Tracks.", 2009. Available on-line: `http://trec.nist.gov/tracks.html` (visited 12/2009).

[OMP09] OpenMP Architecture Review Board OMP. "OpenMP." Available on-line, 2009. `http://openmp.org` (visited 12/2009).

[OPS09] Jie Ouyang, Nilesh Patel, and Ishwar Sethi. "Induction of Multiclass Multifeature Split Decision Trees from Distributed Data." *Pattern Recogn.*, **42**(9):1786–1794, 2009.

[OT08] Nikunj C. Oza and Kagan Tumer. "Classifier Ensembles: Select Real-world Spplications." *Inf. Fusion*, **9**(1):4–20, 2008.

[Par98] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Inc., 1998.

[PBC06] José M. Puche, José M. Benítez, Juan L. Castro, and Carlos J. Mantas. "Fuzzy Pairwise Multiclass Support Vector Machines (4293)." In *MICAI 2006: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pp. 562–571. Springer, 2006.

[Phi09] Phishery. "Phishery – Vendor Independent Information Security Specialists." Available on-line: `http://phishery.internetdefence.net` (visited 12/2009), 2009.

[PLT03] Simon Perkins, Kevin Lacker, James Theiler, Isabelle Guyon, and André Elisseeff. "Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space." *J. Mach. Learn. Res.*, **3**:1333–1356, 2003.

[Pop01] Lubomir Popelinsky. "Combining the Principal Components Method with Different Learning Algorithms." In *IDDM '01: Proceedings of the International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning (ECML / PDKK)*, 2001.

[Pow07] Warren Buckler Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley-Interscience, 1st edition, 2007.

[PP08] Anita Prinzie and Dirk Van den Poel. "Random Forests for Multiclass Classification: Random MultiNomial Logit." *Expert Syst. Appl.*, **34**(3):1721–1732, 2008.

[PT94] Pentti Paatero and Unto Tapper. "Positive Matrix Factorization: A Non-negative Factor Model With Optimal Utilization of Error Estimates of Data Values." *Environmetrics*, **5**(2):111–126, 1994.

[PTR98] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. "Latent Semantic Indexing: A Probabilistic Analysis." In *PODS '98: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 159–168. ACM, 1998.

[PVR03] R. Pearlstein, R. Vaz, and D. Rampe. "Understanding the Structure-Activity Relationship of the Human Ether-a-go-go-Related Gene Cardiac K+ Channel. A Model for Bad Behavior." *J. Med. Chem.*, **46**(11):2017–2022, 2003.

[PY02] Bühlmann Peter and Bin Yu. “Analyzing Bagging.” *Annals of Statistics*, **30**:927–961, 2002.

[Qui93] Ross J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Rak03] Alain Rakotomamonjy. “Variable Selection Using SVM Based Criteria.” *J. Mach. Learn. Res.*, **3**:1357–1370, 2003.

[Rid02] Greg Ridgeway. “Looking for Lumps: Boosting and Bagging for Density Estimation.” *Comput. Stat. Data Anal.*, **38**(4):379–392, 2002.

[Rij79] C. J. van Rijsbergen. *Information retrieval.* Butterworths, 2nd edition, 1979.

[RK03] Marko Robnik-Šikonja and Igor Kononenko. “Theoretical and Empirical Analysis of ReliefF and RReliefF.” *Mach. Learn.*, **53**(1-2):23–69, 2003.

[RM06] Stefan A. Robila and Lukasz G. Maciak. “A Parallel Unmixing Algorithm for Hyperspectral Images.” *SPIE Intelligent Robots and Computer Vision XXIV*, **26**:6384ff, 2006.

[RM07] Stefan A. Robila and Lukasz G. Maciak. “Sequential and Parallel Feature Extraction in Hyperspectral Data Using Nonnegative Matrix Factorization.” In *Systems, Applications and Technology Conference, 2007. LISAT 2007. IEEE Long Island*, pp. 1–7, 2007.

[RM09] Stefan A. Robila and Lukasz G. Maciak. “Considerations on Parallelizing Nonnegative Matrix Factorization for Hyperspectral Data Unmixing.” *Geoscience and Remote Sensing Letters, IEEE*, **6**(1):57–61, 2009.

[Roj96] Raul Rojas. *Neural Networks - A Systematic Introduction.* Springer, 1996.

[RPN06] Niall Rooney, David Patterson, and Chris Nugent. “Pruning Extensions to Stacking.” *Intell. Data Anal.*, **10**(1):47–66, 2006.

[RRB08] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen mei W. Hwu. “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA.” In *PPOPP ’08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 73–82, 2008.

[RS00] Sam T. Roweis and Lawrence K. Saul. “Nonlinear Dimensionality Reduction by Locally Linear Embedding.” *Science*, **290**(5500):2323–2326, 2000.

[RSD07] Cynthia Rudin, Robert E. Schapire, and Ingrid Daubechies. “Precise Statements of Convergence for AdaBoost and arc-gv.” In *Proceedings of AMS-IMS-SIAM Joint Summer Research Conference: Machine Learning, Statistics, and Discovery*, pp. 131–145, 2007.

[RTZ02] O. Roche, G. Trube, J. Zuegge, P. Pflimlin, A. Alanine, and G. Schneider. “A Virtual Screening Method for Prediction of the hERG Potassium Channel Liability of Compound Libraries.” *ChemBioChem*, **3**(5):455–9, 2002.

[RW99] Vijay V. Raghavan and S. K. M. Wong. “A Critical Analysis of Vector Space Model for Information Retrieval.” *Journal of the American Society for Information Science*, **37**(5):279–287, 1999.

[SA06] M. Seierstad and D. K. Agrafiotis. “A QSAR Model of hERG Binding using a Large, Diverse, and Internally Consistent Training Set.” *Chem. Biol. Drug. Des.*, **67**(4):284–96, 2006.

[SBP06] Farial Shahnaz, Michael W. Berry, V. Paul Pauca, and Robert J. Plemmons. “Document Clustering using Nonnegative Matrix Factorization.” *Inf. Process. Manage.*, **42**(2):373–386, 2006.

[SC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[Sch93] Cullen Schaffer. “Overfitting Avoidance as Bias.” *J. Mach. Learn. Res.*, **10**(2):153–178, 1993.

[Sch00] Gisbert Schneider. “Neural Networks are Useful Tools for Drug Design.” *Neural Networks*, **13**:15–16, 2000.

[Sch03] Robert E. Schapire. “The Boosting Approach to Machine Learning: An Overview.” In *Nonlinear Estimation and Classification*, chapter 8, pp. 149–173. Springer, 2003.

[Sch09] Vernon Schryver. “Distributed Checksum Clearinghouses.”, 2009. Available on-line: `http://www.rhyolite.com/dcc/` (visited 12/2009).

[SDH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. “A Bayesian Approach to Filtering Junk E-Mail.” In *AAAI’98: Learning for Text Categorization: Papers from the 1998 Workshop*, 1998.

[SDM06] Claudio De Stefano, Ciro D’Elia, Angelo Marcelli, and Alessandra Scotto di Frecac. “Improving Dynamic Learning Vector Quantization.” In *ICPR ’06: Proceedings of the 18th International Conference on Pattern Recognition*, volume 2, pp. 804–807, 2006.

[Seg04] Mark Segal. “Machine Learning Benchmarks and Random Forest Regression.” Center for Bioinformatics and Molecular Biostatistics, avaialbe from eScholarship; Available on-line: `http://escholarship.org/uc/item/35x3v9t4` (visited 12/2009), 2004.

[Sha51] Claude E. Shannon. “Prediction and Entropy of Printed English.” *The Bell System Technical Journal*, **30**:50–64, 1951.

[Sha03] Peter J. A. Shaw. *Multivariate Statistics for the Environmental Sciences.* Hodder-Arnold, 2003.

[SIL07] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. “A Review of Feature Selection Techniques in Bioinformatics.” *Bioinformatics*, 2007.

[SL98] Rasoul Safavian and David Landgrebe. “A Survey of Decision Tree Classifier Methodology.” *IEEE Trans. Syst. Man Cybern*, **22**:660–674, 1998.

[SLT03] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. “A Classification and Regression Tool for Compound Classification and QSAR Modeling.” *J. Chem. Inf. Comput*, **43**(6):1947–1958, 2003.

[SLT04] Vladimir Svetnik, Andy Liaw, Christopher Tong, and T. Wang. “Application of Breiman’s Random Forest to Modeling Structure-activity Relationships of Pharmaceutical Molecules.” In *MCS ’04: Proceedings of 5th International Workshop on Multiple Classifier Systems*, pp. 334–343, 2004.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., 1986.

[Spa09] Apache Software Foundation. “SpamAssassin Open-Source Spam Filter.”, 2009. Available on-line: `http://spamassassin.apache.org/` (visited 12/2009).

[Spe00] Robert Spence. *Information Visualization.* Addison Wesley, 2000.

[SPE09] SPECS. “The Specs.net Chemistry Database.”, 2009. Available on-line: `http://www.specs.net` (visited 12/2009).

[SPS05] Georgios Sigletos, Georgios Paliouras, Constantine Spyropoulos, and Michalis Hatzopoulos. “Combining Information Extraction Systems Using Voting and Stacked Generalization.” *J. Mach. Learn. Res.*, **6**:1751–1782, 2005.

[SRG03] Ruslan Salakhutdinov, Sam Roweis, and Zoubin Ghahramani. “On the Convergence of Bound Optimization Algorithms.” In *UAI’ 03: Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence*, pp. 509–516. Morgan Kaufmann, 2003.

[SS95] Amit Singhal and Gerard Salton. “Automatic Text Browsing Using Vector Space Model.” In *Proceedings of the Dual-Use Technologies and Applications Conference*, pp. 318–324, 1995.

[SS01] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

[SS04] Alex J. Smola and Bernhard Schölkopf. "A Tutorial on Support Vector Regression." *Statistics and Computing*, **14**(3):199–222, 2004.

[Str07] David J. Stracuzzi. "Randomized Feature Selection." In *Computational Methods of Feature Selection*, pp. 41–62. in Computational Methods of Feature Selection edited by Huan Liu and Hiroshi Motoda, Chapman and Hall/CRC Press, 2007.

[SW07] Olivier Sigaud and Stewart Wilson. "Learning Classifier Systems: A Survey." *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **11**(11):1065–1078, 2007.

[SWY75] Gerard Salton, A. Wong, and C. S. Yang. "A Vector Space Model for Automatic Indexing." *Commun. ACM*, **18**(11):613–620, 1975.

[Sym09] Symantec. "Brightmail: Email Security Appliance, Antispam, Antivirus, Content Filtering.", 2009. Available on-line: `http://www.symantec.com/business/products/family.jsp?familyid=brightmail` (visited 12/2009).

[TC00] Roberto Todeschini and Viviana Consonni. *Handbook of Molecular Descriptors.* John Wiley & Sons, Inc., 2000.

[TE08a] Khac-Minh Thai and Gerhard F. Ecker. "A Binary QSAR Model for Classification of hERG Potassium Channel Blockers." *Bioorg Med Chem.*, **16**(7):4107–4119, 2008.

[TE08b] Khac-Minh Thai and Gerhard F. Ecker. "Classification Models for hERG Inhibitors by Counter-Propagation Neural Networks." *Chem. Biol. Drug. Des.*, **172**:279–289, 2008.

[TGT05] Igor V. Tetko, Johann Gasteiger, Roberto Todeschini, Andrea Mauri, and David Livingstone. "Virtual Computational Chemistry Laboratory - Design and Description." *J. Comput. Aid. Mol. Des*, **19**(6):453–463, 2005.

[TH03] Trevor Tompkins and Dan Handley. "Giving E-mail Back to the Users: Using Digital Signatures to Solve the Spam Problem." *First Monday*, **8**(9), 2003.

[TH04] David A. Turner and Daniel M. Havey. "Controlling Spam through Lightweight Currency." In *Proceedings of the 37th Annual Hawaii International Conference on System Science*, pp. 1–9, 2004.

[TN06] Norikazu Takahashi and Tetsuo Nishi. "Global Convergence of Decomposition Learning Methods for Support Vector Machines." *IEEE Transactions on Neural Networks*, **17**(6):1362–1369, 2006.

[TNN05] Motoi Tobita, Tetsuo Nishikawa, and Renpei Nagashima. "A Discriminant Model Constructed by the Support Vector Machine Method for hERG Potassium Channel Inhibitors." *Bioorg. Med. Chem. Lett.*, **15**(11):2886–90, Jun 2 2005.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison Wesley, 1st edition, 2005.

[TSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science*, pp. 2319–2323, 2000.

[TWM04] Richard Daniel Twining, Matthew M. Williamson, Miranda Mowbray, and Maher Rahmouni. "Email Prioritization: Reducing Delays on Legitimate Mail Caused by Junk Mail." Technical report, HP Laboratories Bristol, 2004. `http://www.hpl.hp.com/techreports/2004/HPL-2004-5.pdf`.

[ULY07] Choong Yong Ung, Hu Li, Chun Wei Yap, and Yu Zong Chen. "In Silico Prediction of Pregnane X Receptor Activators by Machine Learning Approaches." *J. Mol. Pharmacol.*, **71**(1):158–168, 2007.

[Vap99] Vladimir Naoumovitch Vapnik. *The Nature of Statistical Learning.* Springer, 1999.

[VD02] Ricardo Vilalta and Youssef Drissi. "A Perspective View And Survey Of Meta-Learning." *Artificial Intelligence Review*, **18**:77–95, 2002.

[VJ06] P. Viswanath and Karthik Jayasurya. "A Fast and Efficient Ensemble Clustering Method." In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, pp. 720–723. IEEE Computer Society, 2006.

[VK08] Roy J. Vaz and Thomas Klabunde. *Antitargets: Prediction and Prevention of Drug Side Effects.* John Wiley & Sons, Inc., 2008.

[Vos99] Michael D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA., 1999.

[WBS06] Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. "Distance Metric Learning for Large Margin Nearest Neighbor Classification." *Advances in Neural Information Processing Systems*, **18**:1473–1480, 2006.

[WCD03] Stefan M. Wild, James H. Curry, and Anne Dougherty. "Motivating Non-Negative Matrix Factorizations." In *Proceedings of the 8th SIAM Conference on Applied Linear Algebra*, July 2003.

[WCD04] Stefan M. Wild, James H. Curry, and Anne Dougherty. "Improving non-negative matrix factorizations through structured initialization." *Pattern Recognition*, **37**(11):2217–2232, November 2004.

[Wei88] David Weininger. "SMILES – A Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules." *J. Chem. Inf. Comput. Sci.*, **28**(1):31–36, 1988.

[Wei03] Bill Weiman. "AMTP." Available on-line: `http://amtp.bw.org/` (visited 12/2009), 2003.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2nd edition, 2005.

[Wil02] Stefan M. Wild. *"Seeding Non-negative Matrix Factorization with the Spherical K-means Clustering."*. Master's thesis, University of Colorado, 2002.

[Wol92] David H. Wolpert. "Stacked Generalization." *Neural Networks*, **5**:241–259, 1992.

[WTH05] Yu Wang, Igor V. Tetko, Mark A. Hall, Eibe Frank, Axel Facius, Klaus F.X. Mayer, and Hans W. Mewes. "Gene Selection from Microarray Data for Cancer Classification – A Machine Learning Approach." *Comput Biol Chem*, **29**(1):37–46, 2005.

[WWW89] David Weininger, Arthur Weininger, and Joseph L. Weininger. "SMILES 2 – Algorithm for Generation of Unique SMILES Notation." *Journal of Chemical Information and Computer Science*, **29**(2):97–101, 1989.

[WZT04] Jason T. L. Wang, Mohammed Zaki, Hannu T. T. Toivonen, and Dennis E. Shasha. *Data Mining in Bioinformatics.* Springer, 2004.

[XLG03] Wei Xu, Xin Liu, and Yihong Gong. "Document clustering based on non-negative matrix factorization." In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pp. 267–273, 2003.

[XSY04] Y. Xue, L.Z. Sun, Chun Wei Yap, L.Z. Sun, X. Chen, and Yu Zong Chen. "Effect of Molecular Descriptor Feature Selection in Support Vector Machine Classification of Pharmacokinetic and Toxicological Properties of Chemicalagents." *J. Chem. Inf. Comput. Sci.*, **44**(5):1630–1638., 2004.

[XYS04] Y. Xue, Chun Wei Yap, L.Z. Sun, Z.W. Cao, J.F. Wang, and Yu Zong Chen. "Prediction of P-glycoprotein Substrates by a Support Vector Machine Approach." *J. Chem. Inf. Comput. Sci.*, **44**(5):1497–1505., 2004.

[YC05] Chun Wei Yap and Yu Zong Chen. “Prediction of Cytochrome P450 3A4, 2D6, and 2C9 Inhibitors and Substrates by using Support Vector Machines.” *J. Chem. Inf.Model.*, **45**(1):982–992, 2005.

[YD07] Olcay Taner Yildiz and Onur Dikmen. “Parallel Univariate Decision Trees.” *Pattern Recogn. Lett.*, **28**(7):825–832, 2007.

[YL03] Lei Yu and Huan Liu. “Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution.” In *ICML ’03: Proceedings of the 20th International Conference on Machine Leaning*, pp. 856–863, 2003.

[YL04] Lei Yu and Huan Liu. “Efficient Feature Selection via Analysis of Relevance and Redundancy.” *J. Mach. Learn. Res.*, **5**:1205–1224, 2004.

[YM07] Seongwook Youn and Dennis McLeod. “Efficient Spam Email Filtering using Adaptive Ontology.” In *ITNG*, pp. 249–254, 2007.

[YP97] Yiming Yang and Jan O. Pedersen. “A Comparative Study on Feature Selection in Text Categorization.” In *ICML ’97: Proceedings of the 14th International Conference on Machine Learning*, pp. 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[ZBL09] Qiang Zhang, Michael W. Berry, Brian T. Lamb, and Tabitha Samuel. “A Parallel Nonnegative Tensor Factorization Algorithm for Mining Global Climate Data.” In *ICCS ’09: Proceedings of the 9th International Conference on Computational Science*, pp. 405–415, 2009.

[ZH00] Mohammed Zaki and Ching-Tien Ho. *Large-Scale Parallel Data Mining.* Springer, 2000.

[ZTD01] Bernard Zenko, L.jupco Todorovski, and Saso Dzeroski. “A Comparison of Stacking with Meta Decision Trees to Bagging, Boosting, and Stacking with other Methods.” In *IEEE International Conference on Data Mining*, p. 669. IEEE Computer Society, 2001.

[ZZL08] Ping Zhang, Chun-Hou Zheng, Bo Li, and Chang-Gang Wen. “Tumor Classification Using Non-negative Matrix Factorization.” *Communications in Computer and Information Science*, **15**:236–243, 2008.

Biography

Personal	Name: Andreas Janecek
	Born: 22. September 1979, Vienna, Austria
	Nationality: Austria
	Status: unmarried, no children
Affiliation	Research Lab Computational Technologies and Applications Faculty of Computer Science, University of Vienna Lenaugasse 2/8, A-1080 Vienna, Austria
	Phone.: +43 1 4277 39674
	Fax: +43 1 4277 39651
	Email: andreas.janecek@univie.ac.at

Academic Education

actual	Ph.D. Program in Computer Science University of Vienna, anticipated graduation: January 2010
2005	M.S. - Master Studies in Business Informatics University of Vienna, Graduated with distinction
2004	B.S. - Bachelor Studies in Business Informatics University of Vienna

Professional Experience

Since May 2007	Research Associate Research Lab Computational Technologies and Applications University of Vienna `http://rlcta.univie.ac.at`
02/2007-10/2009	Research Associate Department of Distributed and Multimedia Systems University of Vienna `http://www.cs.univie.ac.at`
09/2005-10/2006	Project Assistant "Technology for E-Mail Security" University of Vienna `http://security.ani.univie.ac.at`

Research Interests

Machine learning and data mining methods
(Un)supervised classification and clustering algorithms, feature selection, dimensionality reduction, model evaluation

Low rank approximations
Initialization, parallelization/distribution and high-performance implementations

Application areas
Drug discovery, QSAR-modeling, text mining, e-mail security and classification

December 22, 2009