Efficient Data Mining Algorithms For Time Series and Complex Medical Data

Andrew Zherdin

München 2015

Efficient Data Mining Algorithms For Time Series and Complex Medical Data

Andrew Zherdin

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Andrew Zherdin
aus Kiew

München, den 30.07.2015

Erstgutachter: Christian Böhm

Zweitgutachter: Stefan Conrad

Tag der mündlichen Prüfung: 29.02.2016

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Zherdin, Andriy

Name, Vorname

München, 30.07.2015

Ort, Datum	Unterschrift Doktorand/in

Formular 3.2

Contents

List of Figures

List of Tables

Acknowledgements

Last years I spent writing my PhD Thesis in the Ludwig Maximilian University of Munich was a very important and unforgettable time in my life. Undoubtedly, this work would not have been possible without the support and encouragement of many people. I would like to address my grateful acknowledgement to all of them, even if I unfortunately cannot mention everyone here.

First and foremost, I would specially like to express my sincerest gratitude to my supervisor and first referee on this thesis, Prof. Dr. Christian Böhm, who initiated and supported this work, giving me an opportunity to research on this highly interesting and challenging domain. His outstanding experience, excellent supervision and valuable advices was a subject of inspiration during my research.

Many dearest thanks to Dr. Claudia Plant for her great guidance, fruitful and pleasant cooperation, and friendly encouragement. Furthermore, I warmly thank Prof. Dr. Stefan Conrad and Prof. Dr. Rolf Hennicker for their interest in my work and their willingness to act as the second referee and the Chairman of the Commission on this thesis.

My warmest thanks also go to all my current and past colleagues in the data mining group. In particular, I want to thank Junming Shao, Annika Tonch, Bianca Wackersreuther, Annahita Oswald, Can Altinigneli, Sebastian Goebl, Jing Feng, Xiao He, Nikola Müller, Katrin Haegler, Bettina Konte, Frank Fiedler, Peter Wackersreuther, Michael Plavinski and Son Mai Thai for many inspiring and encouraging discussions and friendly help and support for all these years. In this group I found a very positive and supportive environment giving me much inspiration and freedom for my research.

I am also very grateful to the scientists in the group of Neuroscience of Technical University of Munich, especially to Prof. Dr. Claus Zimmer, Dr. Afra Wohlschläger, Dr. Christian Sorg, Dr. Enrico Schulz, Dr. Christian

Dresel, Dr. Susanne Neufang, Dr. Leonhard Läer, Dr. Velentin Riedl, Dr. Michael Valet, Dr. Andrei Manoliu, Lorenzo Pasquini and Nickolas Myers. Thanks a lot for the pleasant and interesting discussion, cooperation and friendship. I really enjoy working in their enthusiastic and attentive team.

I do not want to miss all the students I had pleasure to supervise during my thesis, who supported my work and who have been beneficial for this thesis. I would like to mention here, Max Liebkies, Robert Noll and Manauel Mesters.

I also would like to sincerely thank the colleagues, who shared their data and source code with me for comparison and analysis. Among them, Xiaozhe Wang, Anthony Wirth, Liang Wang, Mohammed W. Kadous and Ashok Veeraraghavan.

I thank Oleg Sypchenko for comments and critic that greatly improved the thesis.

Finally, special thanks to my parents, my wife Yuliya and my daughters, who are always my powerful backing with their endless love, understanding and support. Without their patience and support, this thesis would not have seen its conclusion.

Abstract

Recently, by means of modern measure approaches and highly developed sensors enormous volumes of data are produced. The data amount is not only huge, but also the data structure is getting more and more complicated. The data originate from the variety of fields such as medicine, research, manufacturing and economy. Together with increasing of data flow, the requirements to algorithms which process these data are increasing too. Queries to databases become more complicated, they concern more and more entities, tolerate less processing time and demand more precise results. The goal of modern Data Mining methods is to manage these challenges.

In this work we are engaged with improvements of algorithms to make them faster and more effective. We introduce algorithms, which solve classical data mining tasks by means of GPUs essentially faster than ordinary one-thread algorithms. This work describes multidimensional index structure that suits very good for graphics processors. We propose an efficient computing of Similarity Join and optimize k-means and DBSCAN for usage on vector processors.

Many modern measure approaches produce time series which are combined into multivariate time series. In such a way, big data volumes are produced, which need very efficient processing. We present in this work algorithms for approximation, classification and clustering, adapted for multidimensional time series and providing precise results without loss of information. Here, the temporal interaction of the time series is used, which has been ignored for other methods. For these purposes we defined a mathematical model based on the set of multidimensional time series. The model takes into account the temporal interaction of individual time series and produces a value, which describes how good an object fits into the model. We introduce methods for aggregating over the length of time series. In such a way, we can search for interesting dependencies and the computing time is independent of the time series length. Classification and clustering decisions can be justified by means of the models, which gives us a powerful tool for data analysis. Especially interesting results were achieved on medical data, since our algorithms are good in treating the fuzziness in complex data. Thus, patterns at interaction between brain regions were found, which are jointly responsible for Alzheimer's, Schizophrenia and Pain-Disorder diseases.

We extend model-based clustering also for hierarchical cluster structure. By means of these extensions, the dependencies between various groups of healthy controls and patients were analyzed and the hypothesis about Alzheimer's disease as aging effect was confirmed. The information concerning the group similarities can be derived from the resulting dendrogram. In this thesis, a workflow for evaluation of pain sensitivity among the healthy persons was proposed. The standard Data Mining methods were here fitted and composed, which allows to forecast the individual pain sensitivity of a certain person based on EEG measurements during laser stimulation.

Zusammenfassung

Mittels moderner Messungsverfahren und hochentwickelter Sensoren werden in der heutigen Zeit riesige Datenmengen produziert. Die Datenmenge ist nicht nur nach dem Volumen groß, sondern auch die Struktur der Daten wird immer komplexer. Die Daten stammen aus verschiedensten Anwendungsbereichen wie z.B. Medizin, Forschung, Produktion und Wirtschaft. Zusammen mit den wachsenden Datenströmen wachsen auch die Anforderungen an die Algorithmen, die diese Daten verarbeiten. Anfragen an Datenbanken werden komplexer, betreffen immer mehr unterschiedliche Datensätze, lassen geringere Antwortzeiten zu und verlangen genauere Ergebnisse. Ein wichtiges Ziel moderner Data-Mining-Methoden ist es, genau diese Herausforderungen zu bewältigen.

In dieser Arbeit befassen wir uns mit verschiedenen Verbesserungen von Algorithmen, um diese schneller und effizienter zu machen. Wir stellen Algorithmen vor, die klassische Data-Miningaufgaben mit Hilfe von Grafikprozessoren wesentlich schneller als klassische Ein-Thread-Algorithmen lösen. In der Arbeit wird eine multidimensionale Indexstruktur beschrieben, die gut für Grafikprozessoren geeignet ist. Wir bieten die effiziente Berechnung des Similarity Join an und optimieren die Clustering-Verfahren k-Means und DBSCAN für den Einsatz auf Grafikprozessoren.

Viele moderne Messverfahren produzieren Daten, die in multivariaten Zeitreihen zusammengefasst werden. So werden große Datenmengen produziert, die eine besonders effiziente Bearbeitung benötigen. Wir stellen in dieser Arbeit Algorithmen für Approximation, Klassifikation und Clustering vor, die auf multidimensionale Zeitreihen angepasst und effizient sind und präzise Ergebnisse liefern, ohne Information aus den Zeitreihen zu verlieren. Dabei wird insbesondere das zeitliche Zusammenspiel der verschiedenen Zeitreihen genutzt, das bei früheren Methoden ignoriert wurde. Für dieses Ziel definierten wir ein mathematisches Modell auf der Menge der multidimensionalen Zeitreihen. Das Modell quantifiziert das zeitliche Zusammenspiel der einzelnen Zeitreihen und liefert einen Wert, der beschreibt, wie gut ein Objekt in das Modell passt. Wir stellen Methoden zum Aggregieren über die Länge der Zeitreihen vor, um interessante Abhängigkeiten zu suchen. Dabei ist die Berechnungszeit von der Länge der Sequenzen unabhängig. Die Entscheidungen bei der Klassifikation und beim Clustering lassen sich anhand der Modelle begründen, wodurch ein leistungsfähiges Instrument zur Datenanalyse entsteht. Besonders interessante Ergebnisse wurden auf medizinischen Daten erreicht, weil unsere Algorithmen gut mit der Ungenauigkeiten in komplexen Daten umgehen können. So wurden Muster bei der Interaktion zwischen Gehirnregionen gefunden, die für Alzheimer, Schizophrenie und eine chronische Schmerz-Krankheit (die somatoforme Schmerzwahrnehmungsstörung) mitverantwortlich sind.

Wir haben das modellbasierte Clustering auch für hierarchische Cluster-Strukturen genutzt. Mit Hilfe dieser Erweiterung wurden Abhängigkeiten zwischen verschiedenen Gruppen von Gesunden und Patienten analysiert und neue Hypothesen über die Alzheimerkrankheit als Alterungserscheinung bestätigt. Die Information zur Gruppenähnlichkeit ist aus dem Dendrogramm abzulesen. Im Rahmen dieser Dissertation wurde auch ein neuer Workflow zur Bewertung der Schmerzempfindlichkeit bei gesunden Personen entwickelt. Hier wurden Standardmethoden der Data-Mining-Forschung angepasst und zusammengesetzt, um aufgrund von EEG-Messungen bei der Laser-Stimulation das Schmerzempfinden einer Person vorherzusagen.

Part I

Preliminaries

Chapter 1

Introduction

We are living in a world, where we have to deal with so large volumes of data, that it is far beyond of our ability to process them manually. New high-capacity storage devices available nowadays, allowing accumulate more and more data, as well as dissemination of digital devices, producing huge amount of raw information (e.g. such as sensors), does not only open new possibilities, but also represent a challenge. This data explosion requires efficient solutions for automated data analysis and information gaining. Knowledge Discovery in Databases (KDD) represents an interdisciplinary research field that focuses on such automatic solutions, laying emphasis on very large data sets.

Section 1.1 in this chapter gives an introduction to the main concepts of the Knowledge Discovery in Databases. The core part of KDD - Data Mining - is presented separately in detail in Section (1.2). Finally, Section 1.3 provides the outline of the thesis, describing how it is organized.

1.1 Knowledge Discovery in Databases

Systematization and unification of the notion of Knowledge Discovery in Databases and the terms related to it go back to 1996. In the paper [48], which can be now considered as a canonical work in KDD and Data Mining field, the authors tried to unify and bring order to a lot of very similar notions for "finding useful patterns in data" existed before, such as information discovery, data pattern processing, knowledge extraction, data mining etc. The result is two basic notions - KDD (Process) and Data Mining, with a clear explanation of distinction between them.

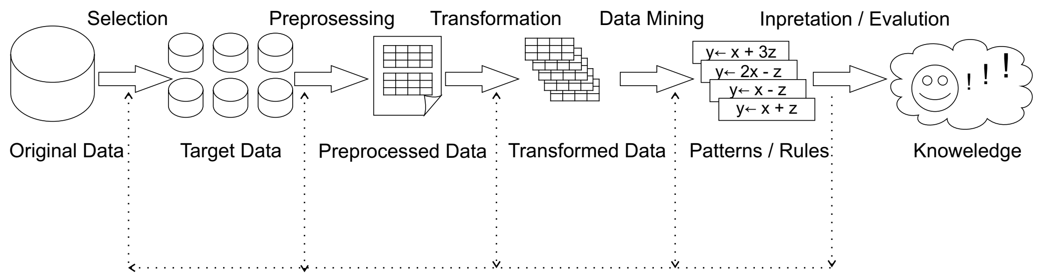


Figure 1.1: The KDD process.

When we speak about Knowledge Discovery in Databases, we lay an emphasis on ”knowledge”, meaning the whole process of ”discovering” new useful ”knowledge”, starting from preparation of given raw data and up to interpretation of extracted patterns. In comparison, Data Mining represents a separate intermediate step in this process, namely the application of specific algorithms for extracting patterns from data. In the following we concentrate us on KDD, whereas the material related to Data Mining we leave for the next Section.

Strictly speaking, Knowledge Discovery in Databases (KDD) is defined as ”the non-trivial process of identifying valid novel, potentially useful, and ultimately understandable patterns in data” [48]. The individual steps of the process are depicted in the Figure 1.1.

Selection: This first step is supposed to create a target set by selecting a data set based on such criteria as type, size, the set of variables etc. Thereby it is focused on data that promise the best performance.

Preprocessing: The purpose of this step is performing cleaning and preparation operations such as noise removal, handling missing data fields as well as accounting additional implicit factors influencing the data.

Transformation: Depending on the task goal, the preprocessed data are reduced at this step by means of projection (dimensionality reduction) and invariant transformation operations. This enables focusing on the particular features of interest, reducing at the same time the effective number of variables.

Data mining: This core step of KDD process is responsible for selecting a data mining method (classification, regression, clustering, etc.), data analyzing and discovery of efficient algorithms, including concrete models and the corresponding parameters, to extract useful patterns in transformed data.

Interpretation and Evaluation: In the last step the extracted patterns are at first supposed to be interpreted and then visualized in a clear understandable form. Finally, the discovered knowledge is incorporated into the existing knowledge system, including reporting the results if needed.

1.2 Data Mining

As already mentioned, Data Mining is a central and in some sense the most interesting part of its big brother KDD. It is not surprising, that these two notions are in certain situations interchangeable and sometimes even used as synonyms. Referred to the formal definition, Data Mining is "a step in the KDD process consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data" [48]. That is, Data Mining is an algorithmic component, focusing on extraction of the patterns from data.

There are two main goals in Data Mining - Verification (checking user's assumption) and Discovery (finding out new patterns). The latter - Discovery - is presented in its turn by predictive tasks, which try to forecast the future behavior based on the current data, and descriptive tasks, presenting the data properties to user in a human-understandable form.

Data Mining methods can be classified as follows:

Classification: Finding a function or developing a model, which is able to assign data to one of several already existing predefined classes. In contrast with clustering, where the clusters (accumulations) need to be identified fist, classification deals with a predefined set of classes. The model can be expressed via mathematical formulae, classification rules, decision trees or neural networks.

Clustering: Detecting a finite set of clusters / categories, similar objects are grouped into. Every such a cluster include data, the most related to each other (w.r.t. a certain similarity measure), whereas data from different clusters are vice versa distinct from each other. At the beginning of the task there is no known class labels.

Outlier Detection: Correct identifying of unconventional objects, so called outliers, which can not be assigned to any of derived clusters or correspondingly to any of predefined classes. Considered by the most of Data Mining methods as noise, detecting outliers can be of great interest in certain tasks, where it is important to discover anomalies in comparison with other objects. Statistical tests, distance measures as well as deviation-based methods can be applied to detect such objects.

Association Analysis: Discovering frequently occurring patterns in data in form of itemsets, sequences or other structures like trees or graphs. Such type of analysis helps to find out important, sometimes not directly visible correlations and associations in data.

Evolution Analysis: Modeling trends and periodicities in objects behavior in the course of time. Beside the other Data Mining methods, also time-series and similarity-based analysis as well as pattern matching approaches find their application here.

Characterization and Discrimination: Associating Data with classes or concepts, which represent a compact summarization of data properties, can be done by means of two approaches - via data characterization, which describes the common properties within a so called target class, or via data discrimination, which compares the general features of the given target class with such from different ones, so called contrasting classes. So, searching for similarities is contrasted here with searching for distinctions.

1.3 Outline of the Thesis

In this thesis, novel algorithms in the areas of clustering, classification and their application in medicine are proposed. Additionally, Data Mining approaches using Graphics Processing Units are presented. The major contributions of this thesis can be summarized as follows:

- optimization of classical Data Mining clustering algorithms for usage in the highly parallel environment of GPU;
- introduction of notion of a mathematical model and its application for solving the tasks of approximation, classification and clustering;
- novel approaches based on attribute selection and classification for confirming hypothesis about different levels of individual sensitivity to pain.

To give a better overview of the work and to ease access to a particular topic let us explain how the further chapters are organized. So, the outline of the thesis looks as follows:

Part I (Chapters 1 to 3) is dedicated to introduction to Data Mining.

Chapter 1 gives a general introduction to KDD and Data Mining.

Next, in **Chapter 2** a brief overview of essential concepts and algorithms, this work refers to, is given.

Chapter 3 presents in a short form the applications of Data Mining in medicine. More detailed information on this subject a reader can find in Chapters 10 and 11.

Part II (Chapters 4 to 6) is dedicated to application of highly parallel algorithms in Data Mining using Graphics Processing Units. Most of these ideas have been published recently ([25, 26]).

Chapter 4 starts with a general introduction to GPU architecture and its programming model. Advantages of using GPU's extreme parallelism compared to CPU are explained.

Chapter 5 is completely devoted to Similarity Join. After introducing the notion of similarity join, the benefits of applying this primitive in Data Mining Algorithms are discussed. Also, a multidimensional index structure optimized for GPU is proposed here. Finally, a GPU-accelerated version of DBSCAN clustering algorithm using Similarity Join is presented [25] and [26].

Further, in **Chapter 6** k-means clustering algorithm specially designed to run on GPU is introduced. The superiority of the approach over its CPU analog [25] is shown.

Part III (Chapters 7 to 9) is dedicated to model-based approaches in Data Mining. Recent publications were used thereby ([106], [24], [155]).

Chapter 7 proposes an intelligent mathematical model-based compression approach to Approximate Clustering of time series. Its experimental evaluation shows very good results compared to existing state-of-the-art approximation methods in terms of clustering accuracy [106].

In **Chapter 8**, a novel model-based classifier is suggested, based on class-specific interaction patterns among time-series. During the evaluation, the approach demonstrates that interaction patterns characterized by linear models are very useful for classification, especially of EEG and fMRI data [24].

Chapter 9 presents an approach for extracting interaction patterns among brain regions. Based on a novel clustering notion, an effective algorithm IKM (interaction K-Means) for partitioning clustering is proposed [155]). The experiments with IKM show especially excellent results on EEG and fMRI data. Moreover, the interaction patterns detected by IKM are easy to interpret and can be visualized. The algorithm is scalable and robust against noise. The paper is extended with an algorithm for searching the number of clusters 9.7.2.

Part IV (Chapters 10 and 11) is dedicated to Data Mining applications in medicine. This part is inspired by the lately published medical papers [169, 149] as well as by some yet unpublished material.

Chapter 10 proposes carrying over the idea of hierarchical subspace clustering to model-based clustering approach applied on the groups of Alzheimer's disease patients and healthy controls. Algorithms show not only interesting results on real-world data, but also a good noise-robustness [149].

Chapter 11 is dedicated to classification of individual sensitivity to pain from the multivariate analysis of Electroencephalography data. An objective neuronal marker of pain sensitivity plays a great role in prevention, diagnosis and treatment of painful conditions [169].

Part V (Chapter 12) concludes the thesis.

Chapter 12 gives a summary of the contributions of this thesis and shows possible directions for future research.

Chapter 2

Main Concepts and Algorithms in Data Mining

2.1 Introduction

In this chapter we briefly consider the principal Data Mining algorithms, this work is based on. In Section 2.2, we shortly describe the fundamental algorithms and data structures, which are used to construct new methods from the following Chapters. Section 2.3 is dedicated to the main classification algorithms. Next, in Section 2.4 an overview of clustering algorithms used in this work are given. In Section 2.5 we consider the fundamental algorithms, which are applied and extended in Chapter 10. The chapter ends with a short conclusion in Section 2.6.

2.2 Similarity-Join, Time Series and Stepwise Selection

2.2.1 Similarity Join

The similarity join represents one of the basic operations of many Data Mining algorithms. As it is clear from the name, its application goal is similarity search which finds its application in Data Mining on feature vectors. An typical initial point in such tasks is a large object set D which one associates with a vector from a multidimensional space, called feature space. The similarity join determines pairs of objects which are similar to each other with

respect to a certain criterion. Choosing a Euclidean distance as such criterion of proximity we have maybe the most well-known kind of similarity join which is the ϵ-join. ϵ-join describes those pairs from $D \times D$ which have a Euclidean distance of no more than a user-defined radius ϵ. If the both points from such a pair are elements of the same set, the join is a similarity self-join. Most algorithms can also be generalized to the more general case of non-self-joins in a straightforward way.
As we claim in [26], it has been shown that the similarity join can be a base for important Data Mining methods such as clustering and classification. Besides, applying a similarity join instead of single similarity queries can accelerate Data Mining algorithms by a high factor. An efficient Similarity-Join on very big volumes of data is described in detail in paper [113].

2.2.2 Time Series

One of the core objects, we work in this thesis with, is a time-series. A time-series represents a sequence of values or events, observed at certain time points, usually with a constant delta between them [72]. A database, dealing with such sequences, are called in the turn time-series database. Time series databases we can meet almost in every application field, whether it be a weather forecasting, science experiment, stock market research or medical diagnostics. Sensors and other intensive data collection tools used nowadays produce a huge amount of time series to be processed. Very often it is desired to process them very fast or even real-time.
Time series analysis has two main goals: modelling and forecasting. The first deals with rules describing the time-series, whereas the latter tries to predict its values in the future.
Time-series modelling consists in its decomposition into so called trend, cyclic, seasonal and random (irregular) components. The original time-series is modelled then as either a sum or a product of these four components. An example of the decomposition of an observed time series into such four parts is depicted in Figure 2.1. The first - trend - represents the main course of the time-series over a long time interval and is typically extracted via moving average (MA) or least squares method. The next one - cycles - determine long-term oscillations (not necessarily periodic) about the trend. The third - seasonal component - refers to very similar to each other deviations systematically happening during some period of the year. Autocorrelation analysis

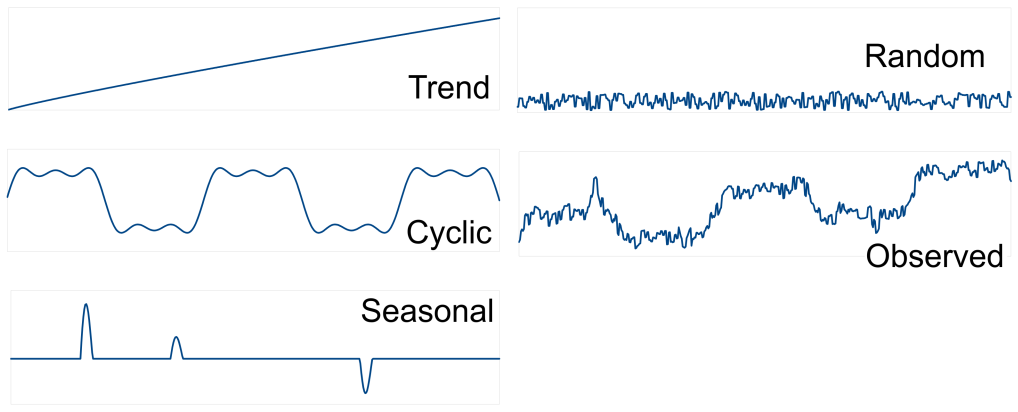


Figure 2.1: Decomposition of an observed time series into four components: Trend, Cyclic, Seasonal and Random.

or seasonal index numbers can be used to detect seasonal patterns. The last part of a time series represents an irregular or random behaviour introduced by the indeterminacy in our world. A detailed introduction to time series analysis and its fundamental methods can be also found in book [29]. ARIMA [29], one more method in Time series analysis, is used for forecasting of its values in the future.
In real-world applications we normally have to deal with multivariate time series, representing multidimensional vector, each component of which is in the turn a one-dimensional time series. That is, for each time point t we have $Y_t = (y_{i_t}, ..., y_{n_t})'$, where each y_{i_t} represents a usual time series. Multivariate time series are very useful when it is needed to describe the interactions between time series variables. For more information on multivariate time-series methods a reader can refer to book [15].

2.2.3 Attribute Selection Algorithms

Before starting processing data it is very important to understand which attributes are not relevant for the given Data Mining task. Often, some of them are also redundant, meaning that they do not contain any useful information any more, since this information is already contained in other attributes.

Not doing this or erroneously dropping the relevant attributes can lead to incorrect results.
The simplest way to choose optimal attributes, under the assumption that they are independent of each other, is to use InfoGain [70] approach. InfoGain is based on the entropy of an attribute.
In the common case, so called attribute subset selection is carried out over the data. Redundant and irrelevant attributes are removed thereby, reducing the original size of data set. As result, a minimal set of arguments is left, with a very similar probability distribution of the data classes compared to such of the original data classes.
Since the brute search over the attributes leads to search among 2^n subsets, heuristic methods are used for finding the appropriate subset. The main such approaches can be classified [72] as:

1. Stepwise forward selection: starting with an empty set of attributes, at each step the best of the not yet chosen attributes is determined and added to the set.

2. Stepwise backward elimination: starting with the full set, at each iteration the worst of the remaining attributes is determined and removed from the set.

3. Combination selection and elimination: alternating application of the both methods above. One starts with two sets - an empty and a full one, chooses the best attribute from the second set and move it to the first set. Then, the worst attribute in the second set is determined and removed. After that, the best attribute in the second is chosen again and moved to the fist one. And so on.

2.3 Classification

The aim of classification is to determine the class of a new object, taking into account the information about known classes, given by the object sets, whose classes we already know. To validate the classification algorithms, x-folds cross-validation method is used. This method consists of the following steps. In the first step, we randomly split the data into x folds, preserving at the same time the proportions of labels in each fold w.r.t their proportions in the full data set. In the second step, the algorithm under test is trained on the data set consisting of $x-1$ folds. The last (not participated for training) fold is used to estimate the accuracy of the algorithm being tested. The second step is then repeated for each fold as test set exactly once. In other words, each object as well as each fold is used once as test set and $x-1$ times as training set.

2.3.1 Next Neighbor Classificator

Despite of the fact, that Next Neighbor Classificator described in[11] is relative simple, it gives often good results on different data. Because of this, $1NN$-Classificator is frequently used as base-line to estimate the quality of new algorithms.
The principle of the classificator is as follows. For the object under test, the closest neighbor from the training set is determined. The class of this neighbor is returned as classification result of the object being tested. There is a simple extension of this method. Instead of searching for one next neighbor, k next neighbors are found. The class of the object under test is in this case the most frequently encountered class label.

2.3.2 Support vector Machines (SVM)

SVM is an interesting technique in classification, suiting for both linear and nonlinear data. The method was originally proposed by Vladimir Naumovich Vapnik in paper [192]. The approach can be described as follows. We try to find an appropriate nonlinear mapping for initial data which maps them into a higher dimensional space. Then it will be searched for an optimal linear hyperplane, separating the data of one class from the data of another. It is known that selecting a sufficient high dimension and suitable nonlinear mapping, it is always possible to find such a hyperplane [72].

The term SVM comes from "support vectors", representing the main training tuples using these support vectors and margins defined by them. The advantage of this method, excluding the benefit that it can be applied on nonlinear data, is that it is very accurate, the trade off however is that it could be very slow. Besides the classification, this method can be used for prediction.

In the case when the data are linear separable, classification test for new tuples can be done by considering such an equation:

$$d(X^T) = \sum_{i=1}^{k} y_i \alpha_i X_i X^T + b_0 \tag{2.1}$$

where X_T is a test tuple, X_i is the i-th support vector (of total number k), y_i is its (support vector's) class label, α_i and b_0 are parameters, determined by SVM algorithms.

Using a given test tuple X_T we can find the resulting sign of the expression above and determine therewith on which side from the hyperplane the test tuple falls. This allows to make a decision concerning the question which class the test tuple belongs to. It is to be noted that the complexity of the classifier here is determined primarily by the number of support vectors rather than by the data dimensionality.

There is an extension of the algorithm for the nonlinear case, i.e. when no hyperplane can be found which separates the data. This extension finds a nonlinear decision for boundaries which allow to classify the test tuples. The algorithm works in two steps: in the first step, the data are transformed (by means of a nonlinear mapping) into a higher dimensional space; in the second step, it is searched for a hyperplane which linearly separates the data in the new higher dimensional space obtained in the first step. The detailed information about nonlinear mapping which can be used for the transformation can be found, for example, in book [72].

2.4 Clustering

The clustering problem consists in partitioning a set of objects into clusters. The objects from the same cluster shall be as similar as possible, whereas the objects from the different ones shall on the contrary be characterized by high dissimilarity.

2.4.1 k-means

One of the most essential clustering methods is the method of k-means, originally proposed by [122]. Assuming the problem of partitioning a set into k clusters, one introduces the proximity to the mean value of the objects

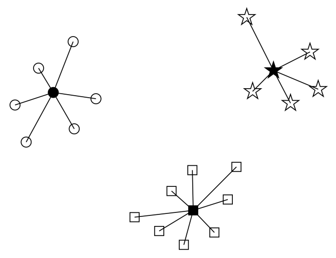


Figure 2.2: Clustering using k-means approach.

as a measure for cluster similarity correspondingly dissimilarity [72]. The problem of finding such centers of gravity is solved iteratively. One starts with a random set of k objects, chosen randomly and which are assumed to be the centroids (means) of the clusters. Having the mean values it is possible to calculate the distance to these k centroids from the other objects to determine the nearest centroid. In such a way, each object is assigned in the turn to its cluster mean, building a cluster. Having k clusters, we act in an opposite way and recalculate the means of each cluster. At second iteration step one repeats the procedure above. And so on. At every step an error of the current partitioning is computed. Usually, the square-error function is taken as such:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2 \tag{2.2}$$

In other word, in each cluster C_i one calculates the squared distances of every object p to its mean m_i and sums them. Minimizing such an error leads to the optimal solution. As a condition to stop the iteration it is sufficient to see that the sequence of errors converges in the limit and the situation that clusters do not change any more is reached.

The complexity of the method is $O(nkt)$, where t is the number of iterations, k is the number of clusters and n is the total number of objects. Under typical conditions, when k and t are significantly less than n, this approach can be considered as effective for large data sets. Usually, t is bounded in our algorithms and therefore the complexity above is simplified to just $O(nk)$. This method has also some disadvantages. First of all, it is needed that the notion of the mean shall be well-defined for the object, which is although very often, but not always the case. Another restriction is that the number

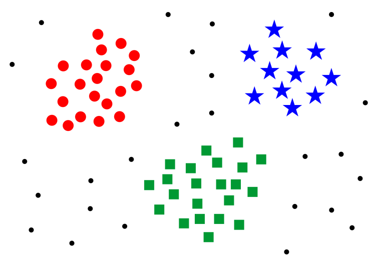


Figure 2.3: Clustering using DBSCAN approach.

of clusters k shall be given in advance. Finally, the method is vulnerable to outliers and noise data.

2.4.2 DBSCAN

Another family of clustering algorithms are density based algorithms. Their main benefit is that they can find clusters of arbitrary shape. The underlying idea is that regions with high object density determine the clusters, whereas the regions with low object density represent the boundaries between them. The base representative of such methods is Density-Based Spatial Clustering of Applications with Noise (DBSCAN)[46]. In this method, a cluster is understood as a maximal set of density-connected objects. A basis for the density-connectivity are so-called core objects, the objects containing in their ϵ-neighbourhood a sufficient (defined in advance) number of objects from a data set D. An object is called directly density-reachable with respect to ϵ from some core object, if the object lies in the ϵ-neighbourhood of this core object. Two points p and q are considered to be density-reachable if it is possible to find a sequence of pairwise directly density-reachable objects $p = p_1, ..., p_n = q$ from D, i.e. such that p_i is directly density-reachable from p_{i+1} for each $1 \leq i \leq n-1$. Finally, an object p is called density-connected to object q in a set D of objects if there exists an further object r, so that both objects p and q are reachable from this object r. All the definitions here are given with respect to threshold ϵ and some minimal number $MinPts$ of points, starting with which an object is considered to be a core one. As result one get a density-based cluster, a maximal set of density-connected objects. The other objects, not belonging to any cluster are viewed as noise.

Assuming n as the number of objects in the set, the complexity of DB-

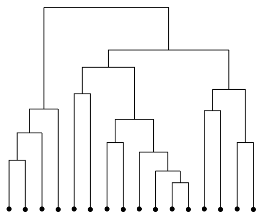


Figure 2.4: Clustering using hierarchical approach.

SCAN algorithm is $O(n^2)$ in general case, and $O(n \log(n))$ in the case of spatial index usage. With reasonable selection of ϵ and the minimal number of objects for core object property, the method can be considered as effective when finding clusters of arbitrary shape.

2.5 Hierarchical Clustering

In comparison to usual clustering, where we just try to determine to which cluster this or that object can be assigned, hierarchical clustering tries to construct a hierarchical decomposition of objects. The latter is done by means of a dendrogram, whose levels partition the datasets into nested clusters.
Hierarchical clustering approaches are presented by agglomerative and divisive methods. The former are characterized by bottom-up approach, starting with its own cluster per each object and iteratively merging these clusters, whereas the latter begins with solely one cluster which is iteratively partitioned into smaller ones.

The steps of an agglomerative method can be described as follows. We start with the set of clusters, which correspond to each object. Then, we merge the two closest clusters to one, which correspond in the turn to a new parent inner node in the dendrogram. Repeating finding closest clusters and merging them, we end finally with only one cluster, forming the dendrogram root. The methods distinguish in the way how the distance between clusters is chosen.

2.5.1 Single- and Average-Link

One of the most basic agglomerative methods is Single-Link [174, 90]. Assumed we are given a distance function between objects, the distance between two disjunct clusters is defined as minimum distance between them, i.e. the smallest distance between all objects pairs, where the first object is taken from one cluster, and the second one respectively from another one.

$$d_{MIN}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

The side effect of Single-Link is a chaining effect, i.e. sometimes the decision to merge the clusters are made only based on individual points which are close to each other and not the whole clusters.

Average-Link [197, 90] uses the average distance between all pairs from two clusters.

$$d_{AVG}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i, y \in C_j} d(x, y)$$

The advantage thereby is that all points are taken into account. This strategy does not lead to chaining effect and besides is not affected by outliers.

2.5.2 Hierarchical Subspace-Clustering

Due of curse of dimensionality, applying traditional clustering algorithms in the case of high-dimensional feature spaces is seriously limited. This happens because different sets of features are relevant for different (subspace) clusters. That is why an extension of traditional clustering called subspace (or sometimes projected) clustering has been developed recently [8, 95, 23, 5]. The aim of subspace clustering is an automatic identifying lower dimensional axis-parallel subspaces of the feature space in which clusters exist. An axis parallel subspace cluster associated with a λ-dimensional projection/subspace, that is a subspace generated by a span of λ attributes, is called a λ-dimensional subspace cluster. In the turn, the dimensionality of a subspace, associated to a subspace cluster, is called subspace dimensionality. There are two classes of subspace clustering algorithms, depending on the resulting types produced by them. The first class represents approaches that allow overlapping clusters, i.e. a point in different projections is permitted to belong to different clusters. The second type, on the contrary, is characterized by the fact that resulting clusters do not overlap, i.e. each point, if it does not represent a noise, may belong exclusively to only one cluster. The algorithms from the first class (allowing overlapping) usually produce a huge number of clusters, which makes interpretation a hard task.

2.6 Conclusion

In this Chapter, a brief overview of main algorithms of classification and clustering was given, which are used in future in this thesis. Also we shortly considered such fundamental concepts as similarity join, dimension selection and time series. In Part III and Part IV, time series are actively applied. Part II proposes a highly parallel algorithm for similarity join. The essential element of the workflow suggested in Part IV is dimension reduction.

Chapter 3

Imaging Modalities and Applications

One of the directions in this thesis is the application of Data-Mining methods in medicine. In this chapter, we consider at first some fundamental tools, being applied in medical diagnostics, such as fMRI and EEG. Applying these methods we research such diseases as Alzheimer's disease, Schizophrenia and Pain Disorder, described in the further sections.

3.1 Functional Magnetic Resonance Imaging (fMRI)

In this section, we take under consideration fMRI, one of the fundamental imaging modalities for researching brain.
Functional magnetic resonance imaging (fMRI) is a non-invasive approach, which indirectly measures the activity of the brain. The local changes in a strong magnetic field, caused by properties of hydrogen nuclear spins, result in capturing of signal changes. How strong these changes are, according to the paper [186], depends on how high the oxygen concentration in blood is. The visualization method using this fact is called blood oxygenation level dependent (BOLD) contrast imaging. Here it is assumed with exceptions as in paper [176], that the brain regions consume at activity more oxygen which can be visualized.
For measuring, a strong magnetic field is needed. The latter (a strong magnetic field) requires much energy and expensive devices. The proband is

placed into a strong magnetic field and the scanner continuously measures the brain activities. The scanning happens in slices, which in 1-3 seconds results in one complete brain image. In accordance with the book [83], the image consists of voxels (3D-pixels). An example for that is presented in the Figure 3.1. The proband should not move during the scanning procedure. The measurements are often accompanied by external stimulations. If no stimulation occurs, one speaks about so called resting state fMRI. Stimulations can be of visual, tactile or pain nature. It is also possible to request a reaction on certain events, for example, pressing a button, hand moving etc. As raw data one gets a series of 3D-images of brain, with voxel sizes of approximately 2-6 mm^3. The first two images should be ignored, since they are not sharp enough. Before the images are evaluated by means of standard methods, they need to be preprocessed. Movement artifacts, such as moving or rotating the head, should be computed and minimized. The images must be zoomed to standard size. The series of images can be also smoothed (both spatially and temporally) and detrended. The preprocessing steps are standard proceedings and are used for noise elimination, whereas they can slightly reduce the sharpness and spatial resolution.

After the measuring and preprocessing one gets a couple of hundreds of 3D-images, approximately 64x64x64 voxels each. Each such a voxel represents a time series. Since the nearby voxels provide very similar information, usually only one time series per region is extracted. For this, the atlases with predefined regions are used. For instance, the authors of [190] divide a brain into 90 regions. Each region is represented by the value, which is the average value of all voxels from the region. In such a way, the data set to process is drastically reduced. For different disorders are different brain regions responsible. So, to keep the data set controllable, the analysis can be focused only on relevant regions. Alternatively, from time series of all voxels only relatively small number of independent components, as in paper [84], can be extracted. In this way, the data set becomes not utopianly huge and needs no additional information in form of an anatomical brain atlas.

In such a way mined data have very good spatial resolution. Since the BOLD signal becomes noticeable only in approximately five seconds and since it is not possible to capture more than one image per second, the temporal resolution is in contrast bad. By means of this method, it is not distinguishable whether a fast happening series of events or a single event takes place. Also high frequency information, according to the authors of [83], is due to its low resolution not contained in the signal.

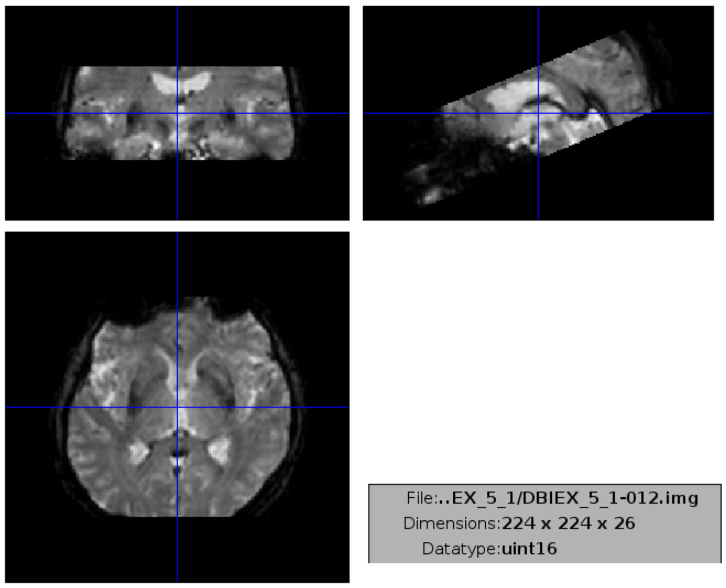


Figure 3.1: In the figure it is depicted one of the frames of fMRI measurements, which consist of many such frames. The light spots in the image mirror the changes in a magnetic field, which correspond to the brain activity in this spot. Although the method allows to scan the whole brain, the scanner is focused here on the given region. In the figure above, the scanner is focused on the part of brain containing the region Hippocampus, which is known to be responsible for AD. In the case of Alzheimer's disease, the changes in this region can be especially clearly seen compared to the images of healthy controls. Due to the decreasing of scanning realm via focusing on only one part of the brain, the spatial scan resolution is essentially increased here.

3.2 Electroencephalography (EEG)

In this section, we give an introduction to relatively simple but very powerful medical diagnostics method.
Electroencephalography (EEG) is a method of medical diagnostics and research, which measures the electrical voltage on the scalp. The potential difference appears as a result of voltage fluctuation in individual brain cells, changing their state on brain activities. Thus, one gets an indirect look at brain activities at different times in the various regions. The method is not invasive.
Electrodes are stuck down in a certain sequence with a special conducting paste on the skin of a proband. The sticking is realized in accordance with standard methods (for example, a 10-20-system, as in book [125], or a uniform distribution over the entire scalp surface), which take into account the various head sizes and forms. Specially prepared flexible headcaps with electrode holders are used here, that adjust themselves to the head size. The electrode measures the potential of nearby neurons (in the range 5-100 μV). For research, as it is stated in [125], 64 electrodes are usually used. The electrode takes spatial information from several square centimeters of the scalp surface. The temporal resolution is essentially better than the spatial one. The signals are recorded with frequencies up to 1 kHz and more as proposed by the authors of [12]. In comparison with fMRI in Section 3.1, according to [74], the method provides more temporal information and less the spatial one. There are also no delays such as BOLD-effect when using fMRI. The reaction on certain stimulation can be seen immediately. The method is relatively simple. No PC-support is needed for recording, since the voltage changes can be recorded on the continuous form paper.
As raw data, one gets per electrode a very long time series with many ten thousands of values. As a rule, only small pieces (intervals) of the time series are taken. More precisely: we are interested only in information, right before and right after the event, e.g. right before and right after a stimulation. There exist several preprocessing methods for raw data. The intervals w.r.t. event can be synchronized and averaged. In such a way, one gets a good temporal resolution, whereas the frequency resolution remains bad. By means of fast Fourier transform (FFT), it is possible to get a very good frequency resolution, however the temporal one tuns out to be bad. With time-frequency analysis it succeeds to get the both resolutions in a good quality. In the latter case, the volumes of data to process increases drastically. Thereby it

can be also observed the increase of redundancy in data.
Different brain activities appear at different frequencies. The authors of [74] distinguish between delta(0-4 Hz), theta (4-7 Hz), alpha(8-12 Hz), beta(13-30Hz) and gamma(30-100 Hz) frequency ranges. The amplitudes in lower frequency ranges are higher, however they tend to be more noised. In the higher frequency ranges, the amplitudes are on the contrary much more smaller but so are also the noise and artifact levels, in accordance with [166]. In different diseases and different kinds of stimulations, these various frequency ranges play an important role.

3.3 Alzheimer's disease

In this section, we shortly describe Alzheimer's disease, for diagnosis of which we apply fMRI method described above.
Alzheimer's disease is a nervous system disease accompanied by large damages of the nerve cells in the brain. The disease happens more likely with age, when one becomes older.
It is characterized by increasing degradation of cognitive performance together with the decrease of daily living activity. Before the first symptoms are noticed, senile plaques as depicted in Figure 3.2 are being formed in the brain of the patient. Senile plaques are microscopic beta-amyloid deposits, which accumulate in the extracellular space of the gray matter [159]. The changes in the brain are incurable.

Alzheimer's disease mostly occurs with increasing of the age. Among 65 year old persons, about two percent show the symptoms of the disease, among the 70 year old it is already three percent and at the age of 75 years it doubles up to six percent. At the age of 85 years, impressive 20 percent are affected according to [19]. Further, the percentage of patients is decreasing, which is explained by the shorter average life expectation of Alzheimer's-diseased patients. After the Alzheimer's diagnosis is established, the remaining average life expectation is about 7 to 10 years in accordance with [132], whereat also large deviations in the both directions are possible. After the beginning of the disease, it can be diagnosed definitely only in several years after that.
The disease passes in four stages. Pre-dementia stage is approximately 8 years before the disease diagnosis can be made for sure. At this stage, the patient has minor short-term memory losses and some difficulties to assim-

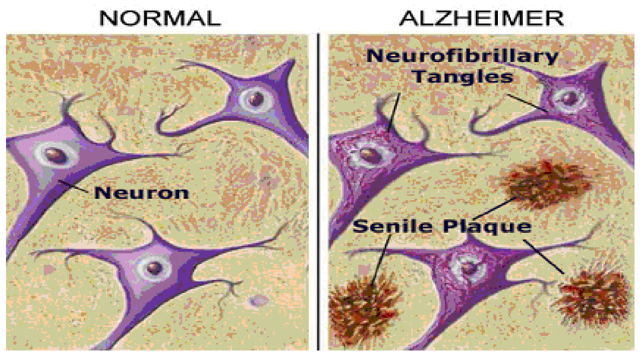


Figure 3.2: Irreversible degenerative changes in the brain of an Alzheimer's disease patient. The protein beta-amyloid deposits of the senile plaques can be observed between the neurons. Illustration from American Health Assistance Foundation [1].

ilate new information according to [165]. Sometimes, slight disturbances in speech comprehension and in trains of thought may be observed. In the early stage the speech disturbances are becoming more distinct in accordance with [164]. Also short-term memory losses are getting noticible. Long-term memory remains however mainly undisturbed, the patients can still continue doing simple daily living activities for their own. In the middle stage, the patient are getting increasing problems with the daily living tasks according to [54]. Also performing fine motor tasks is getting more and more harder for the afflicted person. Coherent speech is becoming for the patient a serious task. At the late stage, the patients do not recognize even long-known persons any more. They can get aggressive or depressive without any essential reason. The patients become often apathetic to their environment. The muscle mass is decreasing, which leads to the other physiological complications, as noted by the authors of [180]. In such a way, the organism is so weakened, that the patients usually die of diseases like pneumonia or heart attack.
The diagnosis of the disease can be made by means of different neurophysiological tests, in accordance with [91, 198]. The last stages of the disease can be observed in the MRT-scanner, since the brain mass decreases, which is clearly seen in the images. The diagnosis at the earlier stages is much more complicated. The essential part of the modern research is an attempt of computer-controlled diagnosis of the Alzheimer's disease using MRT-scanner.

Here it is supposed to use the fact, that also at the earlier disease stages the thinking processes run in the brain in the different way compared to healthy persons. The focus is made on the brain regions, which are affected by the disease.

3.4 Pain disorder

In this section, we give a brief characterization of pain disorder, for diagnosis of which we apply both fMRI and EEG approaches.
Pain is one of the oldest reactions of a living being. Thanks to pain, an organism gets a very clear signal, that it gets hurt and it should response with an escape reflex [82]. Pain is a reaction on a thermal, mechanical or chemical stimulation. When a tissue is damaged, there start running biochemical processes, which stimulate the pain receptors. A quite strong stimulation is needed for excitation of the receptors. A continuing stimulation does not lead to weakening of the excitation. The pain signal is forwarded via nerve fibers. Here, two kind of nerve fibers play an important role. The Fast $A\delta-$fibers conduct the signal at speed approximately 5-30m/s, the slow $C-$fibers at speed 0.5-2m/s. Following the authors of [152] the conducting speed depends upon the stimulated tissue. Pain sensitivity is very subjective among the different people. It is dependent on the environment, duration of the stimulation and pain expectation of a person. In certain situations, such as competition, it could be that people do not feel pain at all. Pain feeling is also possible without any physiological reason for that.
If a person feels for a along time pain without any understandable reason, it is called pain disorder. According to [18], the disease can have one or several following reasons. It can develop after certain pain accompanied diseases as skeletal injuries or some internal organ pathology. A traumatic pain experience, such as car incident or rape can also lead to pain disorder in accordance with [140]. Disturbances in social and job-related fields can accelerate the disease. Emotional pressure as depression and fear accompanies and intensifies it as well. As it is stated in research [50], women are especially prone to pain disorder. Also people in years are more pain sensible than the young ones, as it is observed by the authors of [194]. The pain threshold among the pain disorder patients is considerably reduced. Such people feel pain much more stronger and more disturbing than healthy individuals under similar conditions.

Pain is a very striking event in the brain. It can be easily recognized on EEG and fMRI records. Several pain processed regions of brain are thereby activated. Different persons can evaluate the same stimulation as painful or not. Also the pain intensity and strength is very subjective distinguishable and depends on many environment factors. It is very difficult to determine, if a person indeed has pain disorder or if he or she attempts to cheat. The disease leads to insomnia, depression and work incapacity.
Pain disorder can be treated to help the patients to live their life in a usual way in accordance with the study [18]. Psychosomatic, psychological and psychiatric therapy approaches should be combined thereby, since the disorder can have several reasons simultaneously.

3.5 Schizophrenia

In this section, we give a brief characterization of Schizophrenia, for diagnosis of which we apply fMRI method.
Schizophrenia is a serious mental illness, showing itself via disorders of thought processes, cognition and affectivity. The most prevailing manifestation forms are fundamental disorders in thinking and cognition as well as inadequate or lowered affect.
The symptoms of the disease are divided into positive and negative symptoms [175]. Personality disorders as well as mental delusions up to hallucinations belong to positive symptoms, whereas reducing of facial expression and gesture, affect tailing and short-term thinking belong to the negative ones. The clinical picture is in general very variable, but for a certain person the set of appeared symptoms remains for a long period unchanged. In the case of majoritarian positive symptoms, the patient has a better chance for healing, than in the case when the negative symptoms prevail. The illness can proceed episodically or chronically. After an episode, i.e. after the acute disease phase, the symptoms are abating. Acute phase can last from a couple of weeks up to several months. The next episode can occur in several months or even years. Both adults and children are affected by the disease.
The reasons for schizophrenia consist of many factors. The disease has partially genetic reasons. Certain genes increase the probability of the illness. But genes on their own are not the only reason of schizophrenia. The authors of [124] have shown this with twin research. Less than a half of all pairs of siblings with the same DNA was either simultaneously healthy or simulta-

neously sick. According to the authors of [170], narcotic drugs increase the risk of schizophrenia diagnosis. The life in country decrease the probability to fall ill. Complications during the pregnancy increase the disease risk in accordance with the paper [30].
Schizophrenia is nowadays not really curable. It is however completely possible, to eliminate the disturbing effects of the disease and to supply a normal life for the patients. Primarily, medicamentous treatment is thereby applied. It helps mainly in the acute phase of disease against the positive symptoms. Some medicaments can however enforce the negative symptoms. Social binding correspondingly social therapy together with occupational therapy are also very helpful and give a positive perspective. Sport activities are helping as well and show a positive effect.
According to the paper [43], among schizophrenia patients it is usually observed less functional difference between brain hemispheres than among healthy persons. Such a difference should be clearly seen with fMRT. The dependencies here are not trivial and show themselves only time-delayed. This kind of dependencies with time delays are called Granger causality. The objective of the modern research is finding method, allowing automatically to determine the diagnosis (as early and exactly as possible) based on results of imaging approaches such as fMRT.

In this chapter, we considered such diagnostics tools as fMRI and EEG, which we applied to diagnose such illnesses as Schizophrenia, Alzheimer's disease and pain disorder. However, very often in medical application we deal with huge amount of data which can represent a challenge even for modern computers equipped with newest CPUs. Using alternative computing power in form of Graphical Processing Units, which are able to highly parallelize the calculations, is very effective, efficient and promising direction. For example, to accelerate the computations of SVM in paper [169], we successfully used the power of graphic processors. More about GPU computing can be found in the next Chapters.

Part II

Data Mining Using Graphics Processing Units

Chapter 4

Parallel Computing using GPU

Within the last several years, Graphics Processing Units (GPU) have evolved from special-purpose devices for the display signal preparation, supporting typical computer graphics tasks such as rendering of 3D scenarios, into powerful coprocessors that can just as well be used for general numeric and symbolic computation tasks like simulation and optimization. As chief benefit, GPUs supply extreme parallelism (with hundreds simple programmable processors) coupled with a high memory bandwidth at comparably low cost. In this chapter, we give a brief characterization of GPU programming model and its features.
Parts of the material presented in this chapter have been published in Paper [25]. In this paper, Andrew Zherdin has codesigned the main concepts for parallelization as well as implemented and carried out a part of experiments. Robert Noll has implemented and optimized the most of source code. His task was also the choice and installation of the used hardware. Bianca Wackersreuther has taken an important part in the development and optimization of the algorithms and experiments. Christian Böhm and Claudia Plant have supervised this work and essentially improved its results.

4.1 Introduction

In recent years, *Graphics Processing Units* (GPUs) have evolved from special-purpose devices for the display signal preparation into powerful coprocessors supporting the CPU in different ways. Graphics applications such as modern realistic 3D games are extremely computationally demanding and require a huge number of sophisticated algebraic operations to update the display image (called frame). For supplying such applications with an appropriate computing power, modern graphics hardware includes a large number of programmable processors which are optimized to cope with this high workload of vector, matrix, and symbolic computations in a highly parallel way. Although GPU and CPU are not directly comparable due to their different architectures, speaking in terms of peak performance, the GPUs have surpassed state-of-the-art multi-core CPUs by a large margin.

To neutralize the exponential data growth (according to researches, the amount of scientific data is roughly doubling every year [184].), in many research communities such as mechanical simulation [185], cryptographic computing [20], life sciences [119, 126], or machine learning [37] there is a great aspiration to apply the outstanding computational capabilities of GPUs also for another purposes not at all related to computer graphics. The corresponding research area is called therefore General Purpose Computation on Graphics Processing Units (GP-GPU).

The actively developing research area of Data Mining proposes methods providing the transformation of the raw data into profitable knowledge. The main Data Mining tasks involve classification, clustering, regression, outlier identification, as well as frequent itemset and association rule mining, which find scientific and commercial application in neuroscience, astronomy, biology, marketing, and fraud detection. Data Mining tasks are divided into supervised and unsupervised ones. Classification and regression are supervised Data Mining tasks, since their goal is to examine a model for predicting a predefined variable. The other tasks are unsupervised, because the user does not identify a priori any of the variables to be learned. Instead, the algorithms themselves have to pick out any interesting patterns and regularities in the data.

We concentrate in this thesis on NVIDIA's technology *Compute Unified Device Architecture* (CUDA) [2]. Recently, anticipating the interest towards general purpose computing on GPU, graphics hardware vendors developed libraries, pre-compilers and application programming interfaces to support GP-GPU applications. CUDA offers an API for the C programming language, by which both the *host program*, being the main program, as well as the so-called *kernel functions* are assembled in a single program [2]. The host program is executed on the CPU, whereas *kernel functions* are executed in a highly parallel art on the (hundreds of) GPU processors. ATI (now subdivision of AMD), the only real NVIDIA's rival now, offers a similar technique using the brand names Close-to-Metal, Stream SDK, and Brook-GP.

The remainder of this chapter is organized as follows: Section 4.2 represents a general review of the related work. Section 4.3 gives an introduction to the graphics hardware and explains the CUDA programming model. Section 4.4 summarizes the chapter.

4.2 Related Work

In this section, we provide a review of the related work in GPU processing.

General Processing-Graphics Processing Units. Beside its primary task in form of graphics processing, theoretically, GPU is capable of performing any computation. Important is, that this computation can be derived to the model of parallelism and is suitable for the specific architecture of the GPU. In many research areas, this opportunity has been examined and very promising results have been achieved. In the field of life sciences, a new technique for high performance molecular dynamics simulations on GPUs, designed and implemented using CUDA, was introduced in [119] by Liu et al. Their results show a significant performance gain on an NVIDIA GeForce 8800 GTX graphics card compared to sequential processing on CPU. In another paper, Manavski and Valle [126] suggest an extremely fast realization of the Smith-Waterman algorithm, a procedure for similarities search in protein and DNA databases, running on GPU and implemented using the CUDA platform. Significant speedups are achieved on a workstation running two GeForce 8800 GTX.

Mechanical simulation is one more widespread application area that take advantage from the high processing power of the GPU. For example, Tascora et al. [185], propose a novel approach to solve large cone complementarity problems by means of a fixed-point iteration algorithm, in the context of simulating the frictional contact dynamics of large systems of rigid bodies. Same as the above-mentioned methods in the area of life sciences, the algorithm is also implemented using CUDA for a GeForce 8800 GTX to simulate the dynamics of complex systems.

We can continue for a long time to give examples concerning the possibilities of performing computations on the GPU. It is clear, that there are nearly boundless such applications. So, we just mention one more example from another popular field, where parallel computations on a GPU can be in

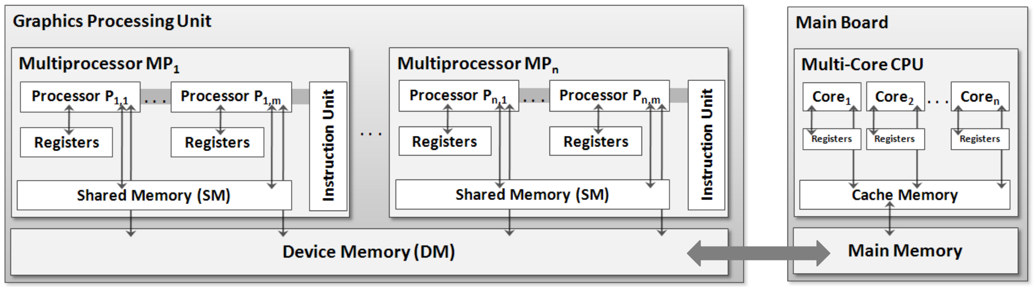


Figure 4.1: Architecture of a GPU [25].

a great demand. We speak about cryptographic computing. In paper [20], the authors present realization of the elliptic curve method (ECM) of integer factorization, showing an outstanding performance. The record-breaking speeding-up profits from two NVIDIA GTX 295 graphics cards, using a new ECM implementation relying on CUDA's new parallel addition formulas and functions.

4.3 Architecture of the GPU

Today's Graphics Processing Units (GPUs) are powerful computing coprocessors, capable of supporting the Central Processing Unit in many different ways, not only in typical for them tasks like interactive games, CAD and 3D-modeling applications, but also for general-purpose computing (in this case, we call them GP-GPUs). From the hardware point of view, a GPU represents a chip including a number of multiprocessors. Each of these multiprocessors consists in its turn of a set of simple SIMD processors, i.e. all processors of one multiprocessor execute in a synchronized way the same arithmetic or logic operation simultaneously, potentially operating on different data. For instance, the GPU of the newest generation GT200 (e.g. on the graphics card Geforce GTX280) has 30 multiprocessors, each consisting of 8 SIMD-processors, summarizing to a total amount of 240 processors inside one GPU. The computational power sums up to a peak performance of 933 GFLOP/s.

4.3.1 The Memory Model

As it is shown in Figure 4.1, besides special purpose memory in the context of graphics processing (e.g. texture memory), there are three significant types of memory, relevant for GPU: shared memory (SM), device memory (DM) and main memory. The *shared memory* (SM) is the fastest but at the same time the most expensive one, and as consequence, is very limited in size (about 16 KBytes per multiprocessor). Physically, it is situated inside of a multiprocessor. This memory is shared among all processors within one multiprocessor and can be accessed at the speed of register, i.e. has no delay. Typically, this kind of memory is used for local variables and for communication between the threads of the processors of the same multiprocessor. Each multiprocessor has its own SM, so the data exchange between the threads of the processors from different multiprocessors is impossible.

The *device memory* (DM), the second type of memory GPU works with, is the actual video RAM of the graphics card (as well used for frame buffers etc.). Physically, DM is located on the graphics card, but not inside the GPU itself. Compared to SM, it is much larger (up to some hundreds MBytes or even a few Gbytes), but also significantly slower. In particular, memory accesses to DM yield a typical latency delay of 400-600 clock cycles (on G200-GPU, corresponding to 300-500ns). The bandwidth for transferring data between DM and GPU (141.7 GB/s on G200) is however essentially higher than that of CPU and main memory (about 10 GB/s on current CPUs). DM can be used to share information between threads on different multiprocessors. If some threads schedule memory accesses from contiguous addresses, these accesses can be coalesced, i.e. taken together to improve the access speed. A typical cooperation pattern for DM and SM is to copy the required information from DM to SM simultaneously from different threads (if possible, considering coalesced accesses), then to let each thread compute the result on SM, and finally, to copy the result back to DM.

The third and the last type of memory considered here is the *main memory*. It is located on the mainboard, outside of the graphics card. Whereas the GPU has no access to the address space of the CPU, the CPU can write to or read from DM only by means of specialized API functions. The disadvantage in this case is however, that the data have to be transferred via the Front Side Bus and the PCI-Express Bus. The bandwidth of these bus systems is strictly limited, and therefore, these special transfer operations are considerably more expensive than direct accesses of the GPU to DM or direct accesses of the CPU to main memory.

4.3.2 The Programming Model

The central concept in the GPU programming model are threads, which can be considered as lightweight processes, well suited for creating and synchronizing. Unlike CPU processes, generation and termination of GPU threads as well as context switches between them do not lead to any significant overhead. In GPU-based applications, we are dealing with thousands or even millions of threads. In 3D games, for example, one thread per pixel can be created. To avoid the latency delay of memory accesses and increase by that the efficiency, it makes sense to create even much more threads than the number of physical SIMD-processors in GPU. In particular, as already mentioned, accessing the DM can cause a latency delay up to 400-600 clock cycles, during which the context switch to another threads can help feeding the multiprocessor with new portions of work, preventing it thereby from idle running. Threads on GPU, each of which executes so-called *kernel function*, can be created via API calls provided by the CUDA programming library. The host program, which is executed sequentially on the CPU, together with the kernel functions, which are executed concurrently on the GPU, are defined in a superset of the C programming language. Certain functionalities are however restricted when using the kernel functions (e.g., no recursion).

Another important notion in the GPU programming model are so-called *warps*. GPU threads do not have an individual instruction pointer, an instruction pointer is instead shared by several threads, grouped into a *warp* (usually 32 threads per warp). Each warp is processed simultaneously on the 8 processors of a single multiprocessor (SIMD) using 4-fold pipelining (as result, 32 threads are executed fully synchronously). It can be, that not all the threads follow the same execution path. In this case, the different execution paths are executed in a serialized way. The number of SIMD-processors per multiprocessor (8) and 4-fold pipelining concept are constant on all current CUDA-capable GPUs.

Multiple warps are grouped in turn into *thread groups* (TG). It is recommended [2] that the total number of threads in a TG is a multiple of 64. The different warps (both in the same and in different TGs) are executed independently. Unlike the threads from different TGs, the threads in one TG use the same shared memory, which allows them to communicate and share the data via SM. The threads in the same thread group can be synchronized (let all threads wait until all warps of the same group have reached that point of execution). The latency delay caused by DM access initiated by some warp can be minimized by switching to another warp of the same or a different thread group. To avoid overhead from switching between warps of different TGs on a multiprocessor, NVIDIA recommends that each thread uses only a small fraction of SM and registers of the multiprocessor [2].

4.3.3 Atomic Operations

To synchronize parallel processes and to assure the correctness of parallel algorithms, CUDA offers a large number of various *atomic operations*, among which are also increment, decrement and exchange. The latter ones we will need for our algorithms. Most of the atomic operations are applied on integer data types in Device Memory. The newest version of CUDA (Compute Capability 1.3 of the GPU GT200) supports also atomic operations in SM.

For example, if there is a list as a common resource with concurrent reading and writing from/to it, shared by some parallel processes, it may be necessary to (atomically) increment a counter for the number of list entries (which is in most cases also used as the pointer to the first free element of the list). The fact, that operation is atomic, implies in this case the following two requirements: If two or more threads increment the list counter, then (1) the value counter after all parallel increments must be equivalent to the value before plus the number of these parallel increment operations. And, (2), each of the concurrent threads must obtain a separate result of the increment operation which indicates the index of the empty list element to which the thread can write its information. Therefore, most atomic operations return a result after their execution. Let us consider as example the `atomicInc` operation. It has two parameters, one of which is address of the counter to be incremented, and another (optional) one is a threshold value which must not be exceeded (as result of the operation). When applying the operation, the following occurs: The counter value at the indicated address is read and incremented (ensuring that the threshold is not exceeded). The kernel method, invoked the `atomicInc`, gets the *old* value of the counter (before incrementing) as return value. If it happens, that several threads (of the same or different thread groups) call some atomic operations simultaneously, the result of these operations is that of an arbitrary sequential applying of the concurrent operations. The `atomicDec` operation works analogously.

The `atomicCAS`, performing a Compare-and-Swap operation, has three parameters, an address, a compare value and a swap value. If the value at the address equals the compare value, the value at the address is replaced by the swap value. Independently of the case, the invoking kernel method receives as return value the old value at the address (before swapping).

4.4 Conclusions

In this chapter, we demonstrated how Graphics processing Units (GPU) can effectively support highly complex computational tasks. Algorithms, customized to the special environment of the GPU are characterized by extreme parallelism at low cost. Going beyond the primary scope of this Chapter, these building blocks are applicable to support a wide range of Data Mining tasks, including outlier detection, association rule mining and classification.

Chapter 5

Similarity Join Based Methods using GPU

In this chapter we proceed to the application of GPU-based computing and propose several algorithms for such computationally expensive Data Mining tasks as similarity search and clustering, specially designed for the highly parallel GPU environment. We define a multidimensional index structure specifically suited to support similarity queries under the restrictions of GPU programming model, and define a similarity join method. Additionally, we introduce a highly parallel algorithm for density-based clustering.
The concepts described in this chapter have been published in Papers [25, 26]. The paper [25] is an extended version of the paper [26]. In this paper, Andrew Zherdin has codesigned the main concepts for parallelization as well as implemented and carried out a part of experiments. Robert Noll has implemented and optimized the most of source code. His task was also the choice and installation of the used hardware. Bianca Wackersreuther has taken an important part in the development and optimization of the algorithms and experiments. Christian Böhm and Claudia Plant have supervised this work and essentially improved its results.

5.1 Introduction

As we already noted in the last Chapter, *Graphics Processing Units* (GPUs) have evolved recently from special-purpose devices to powerful coprocessors allowing to improve dramatically the computation of many tasks.

In this chapter, we concentrate on application possibilities of using the computational power of GPUs for *Data Mining* tasks. Data Mining consists of "applying data analysis algorithms, which, under acceptable efficiency limitations, produce a particular enumeration of patterns over the data" [48]. It is important to note, that the exponential increase in data is not necessarily accompanied by a correspondingly large gain in knowledge.

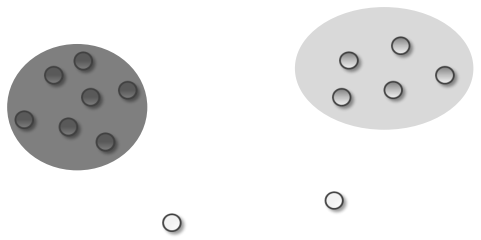


Figure 5.1: Example for Clustering.

Clustering is probably the most common unsupervised Data Mining task. The aim of clustering is to find a natural grouping of a data set. With other words, data objects assigned to a common group (called cluster) are supposed to be as similar as possible, whereas objects assigned to different clusters are expected to differ as much as possible. For instance, consider the set of objects shown in Figure 5.1. A natural grouping would divide the objects into two different clusters. Two objects not fitting well to any of the clusters are considered as outliers and should be left unassigned. To introduce the definition of clustering, we need to specify the notion of similarity among objects. In most cases, the similarity is expressed in a vector space, called the feature space. In Figure 5.1, we indicate the similarity among objects by representing each object by a vector in two dimensional feature space.

In general case, the objects are characterized by a certain number d of numerical properties (from a continuous space) which are extracted from the objects and put together to a vector $x \in \mathbb{R}^d$. Thus, the number of properties d which have been extracted defines the dimensionality of the feature space. Let us consider a concrete case, where as objects we take a set of orchids. The phenotype of orchids can be characterized by means of the lengths and the widths of the two petal and the three sepal leaves, of the form (curvature) of the labellum, and of the colors of the different compartments. In the example shown in Figure 5.2, 5 features are measured, and each object

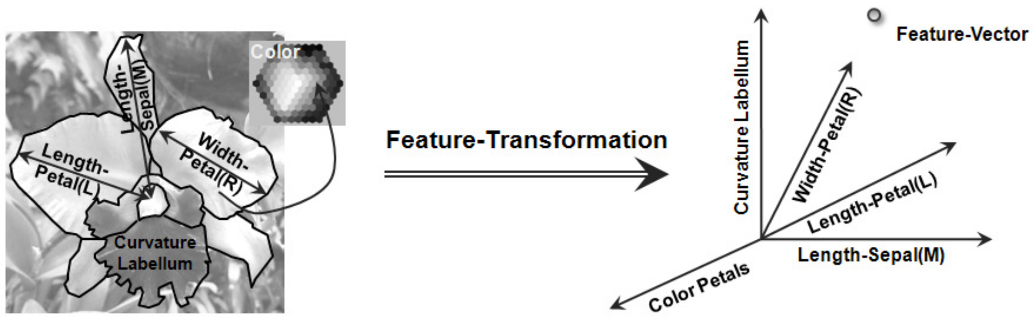


Figure 5.2: The Feature Transformation.

is thus transformed into a 5-dimensional vector space. Usually, to measure if one feature vector is similar to another, a distance function like the Euclidean metric is used. In real-life applications, where the objects are kept in a large database, for typical similarity searching tasks (like to find a number k of nearest neighbors, or objects having a distance that does not exceed a threshold ϵ) normally *multidimensional index structures* are used. The search efficiency is achieved in this case by means of a hierarchical organization of the data set. The well-known indexing methods (e.g. the R-tree [66]) are designed and optimized for secondary storage (hard disks) or for main memory. The usage of GPU, due to its highly parallel but restricted programming environment, supposes developing specialized indexing approaches. In this chapter, we propose such an indexing method.

As it has been shown, in clustering and many other Data Mining algorithms, a powerful database primitive can be very useful: The *similarity join* [22]. The result of applying this operator are all pairs of objects in the database, the distance between which is less than a certain predefined threshold ϵ. For the similarity join we propose two specially GPU-based algorithms, one being a nested block loop join, and one being an indexed loop join, using the indexing structure mentioned above.

Finally, to demonstrate that also highly complex Data Mining tasks can be efficiently implemented using novel parallel algorithms, we propose a parallel version of the widespread clustering algorithm: the density-based clustering algorithm. We show how the density-based clustering algorithm DBSCAN can effectively profit from the parallel similarity join. In the next chapter, we present a parallel version K-means clustering, which follows an algorithmic paradigm that is very different from density-based clustering.

When implementing algorithms for GPU, we used NVIDIA's technology *Compute Unified Device Architecture* (CUDA) [2].

The remainder of this chapter is organized as follows: Section 5.2 represents a general review of the related work in GPU processing with focus on database management and Data Mining. Section 5.3 proposes a multi-dimensional index structure for similarity queries on the GPU. Section 5.4 presents the GPU-based implementations of non-indexed and indexed join. Section 5.5 is dedicated to GPU-capable algorithm for density-based clustering. Section 5.6 presents the results of an extensive experimental evaluation of our approaches, and Section 5.7 summarizes the chapter and suggests directions for future research.

5.2 Related Work

In this section, we provide a review of the related work in GPU processing with focus on database management and Data Mining.

Acceleration of relational database operations on GPU is also an important research area and the one which we interest in. Various techniques in this field are proposed by some papers.

For instance, paper [78] introduces some algorithms for the relational join on an NVIDIA G80 GPU by means of CUDA. Two recent papers [115, 26] are dedicated to the topic of similarity join in feature space. Similarity join yields all pairs of objects from two different sets R and S fulfilling a certain join predicate. Usually, the role of join predicate plays the ϵ-join which determines all pairs of objects having a distance of less than a predefined threshold ϵ.

An algorithm based on the concept of space filling curves, e.g. the z-order, for pruning of the search space, is proposed by the authors of [115]. It uses the CUDA platform and runs on an NVIDIA GeForce 8800 GTX graphics card. Due to highly parallelized sorting, the z-order of a set of objects suits very good to be computed on GPU. For efficient pruning, their algorithm operates on a set of z-lists of different granularity. On the other hand, the algorithm has its disadvantages. First of all, in higher dimensions one has degrading of performance, since all dimensions are treated equally. Furthermore, because of uniform space partitioning in all data space areas, space filling curves can not be applied for clustered data. In the second paper [26], we propose a method that overcomes that kind of problem. Their solution is parallelizing the baseline technique, which underlies any join operation with an arbitrary join predicate. More precisely, the authors used the parallel version of the nested loop join (NLJ), a powerful database primitive that can find many applications, including Data Mining. The experiments of their CUDA-based implementation are carried out on NVIDIA 8500GT graphics card.

Govindaraju et al. [59, 60] show that operations, significant for query processing in databases, like sorting, aggregations, conjunctive selections, and semi-linear queries can be essentially accelerated when using GPUs.

5.3 An Index Structure for Similarity Queries on GPU

Similarity queries are frequently used as an important building block by many algorithms, solving typical Data Mining problems like classification, clustering, outlier detection and regression. Being often the major part of computation in the corresponding Data Mining tasks, it is very significant thus to treat similarity queries with highest efficiency. We define similarity queries as follows: Given is a database $\mathcal{D} = \{x_1, ...x_n\} \subseteq \mathbb{R}^d$ of a number n of vectors from a d-dimensional space, and a query object $q \in \mathbb{R}^d$. The following two types of similarity queries are distinguished, the range queries and the nearest neighbor-queries:

Definition 1 (Range Query) *Let $\epsilon \in \mathbb{R}_0^+$ be a threshold value. The result of the range query is the set of the following objects:*

$$N_\epsilon(q) = \{x \in \mathcal{D} : \quad ||x - q|| \leq \epsilon\}.$$

where $||x - q||$ is an arbitrary distance function between two feature vectors x and q, e.g. the Euclidean distance.

Definition 2 (Nearest Neighbor Query) *The result of a nearest neighbor query is the set:*

$$NN(q) = \{x \in \mathcal{D} : \quad \forall x' \in \mathcal{D} : \quad ||x - q|| \leq ||x' - q||\}.$$

Definition 2 is as well generalizable for the case of the k-nearest neighbor query ($NN_k(q)$), where a number k of nearest neighbors of the query object q is retrieved.

The availability of multidimensional index structure supporting the similarity search can dramatically improve the performance of similarity queries. For this purpose, kernel functions can be used, which make possible to traverse our index structure for many search objects concurrently. As we remember, kernel functions have some restrictions when using them, among which the absence of recursion, requirement of a small storage overhead by local variables etc. For this, we must keep the index structure as simple as possible. To provide a good compromise between selectivity of the index and its simplicity, we propose a data partitioning method using a constant number of directory levels. Each level is supposed to partition the data set $\mathcal{D}$ in accordance with its dimension. We start with the first level, which divides the data set according to the first dimension of the data space, the second level according to the second dimension, and so on. To achieve good results using this approach, the assumption about a high selectivity in the first dimensions is needed. This implies, that certain transformations should be applied before starting the actual Data Mining method (e.g. Principal Component Analysis, Fast Fourier Transform, Discrete Wavelet Transform, etc.). Parallelization of these transformations by means of GPU is possible, but it was not considered as a part of this work.

A simple example of a 2-level directory (corresponds to 2-dimensional case) is given in Figure 5.3, which is similar to [103, 112]. The root node in the figure is considered as level-0. In our experiments in Section 5.6, we used an advanced 3-level directory with fanout 16.

The thing we need to do before starting the Data Mining task itself, is to construct our simple index. It is done in a bottom-up way by fractionated sorting of the data: At first, we sort the data set according to the first dimension and partition it into the specified number of quantile partitions. After that, we sort separately each of the partitions according to the second dimension, and so on. As data type to store the boundaries we take simple arrays, allowing easy access in the subsequent kernel functions.

As we can see, our index construction can already be done on the GPU, because it is based on sorting, for which efficient GPU-based sorting methods have been proposed [59]. Since bottom up index construction is usually not very expensive compared to the Data Mining algorithm, our approach performs this preprocessing step on CPU.

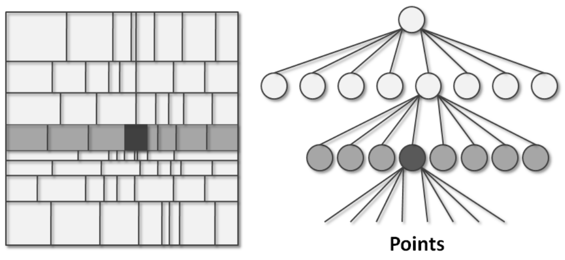


Figure 5.3: Index Structure for GPU.

In the initialization phase of the Data Mining method, we need to transfer the data set from the main memory into the device memory of graphics card. Besides this action, also the directory (in other words, the arrays containing the coordinates of the page boundaries as described above) must be transferred. The directory size is always small compared to the complete data set, thus its overhead can be practically ignored.

The major modification, we make in the kernel functions in our Data Mining methods, concerns the determination of the ϵ-neighborhood of some given seed object q. This determination is efficiently done by means of SIMD-parallelism inside a multiprocessor but in a slightly different ways, depending if we are dealing with index or non-indexed version. In the latter case (non-indexed version), the determination is done by a set of threads (within a thread group). Thereby, each of these threads makes iteration over a different part of the (complete) data set. In the former case (indexed version), one of the threads iterates in a set of nested loops (each level of the directory corresponds to one loop) over those nodes of the index structure which represent regions of the data space which are intersected by the neighborhood-sphere of $N_\epsilon(q)$. In the innermost loop, it remains one set of points (corresponding to a data page of the index structure). This set is then processed by means of the SIMD-parallelism, in a similar manner to the non-indexed version.

5.4 The Similarity Join

The *similarity join* represents a basic operation of a database system. It is designed for such tasks as similarity search and Data Mining on feature vectors. In such applications we are dealing with a database $\mathcal{D}$ of objects, for each of which a vector from a multidimensional space can be assigned. This multidimensional space is accordingly called the feature space. The similarity join determines the pairs of objects which are similar to each other, with respect to the metrics chosen in the feature space. The most known and widespread form of the similarity join is the ϵ-join yielding those pairs from $\mathcal{D} \times \mathcal{D}$, a Euclidean distance between which is no more than a user-predefined threshold ϵ:

Definition 3 (Similarity Join) *Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a set of feature vectors of a d-dimensional vector space and $\epsilon \in \mathbb{R}_0^+$ be a threshold. Then the similarity join is the following set of pairs:*

$$SimJoin(\mathcal{D}, \epsilon) = \{(x, x') \in (\mathcal{D} \times \mathcal{D}) : \quad ||x - x'|| \leq \epsilon\},$$

If x and x' are elements of the same set, we are dealing with a similarity self-join. The method proposed in this chapter (as well as most algorithms) can also be extended to the more general case of non-self-joins in a straightforward way. For this moment, also algorithms for a similarity join with nearest neighbor predicates have been proposed.

To summarize, we note that the similarity join represents a powerful building block for similarity search and Data Mining tasks. It has been shown that important Data Mining methods such as clustering and classification can be based on the similarity join. Proposing effective algorithms for similarity join we propose effective solutions for the mentioned problems. Using a similarity join instead of single similarity queries can lead to speeding up Data Mining algorithms by a high factor [22].

5.4.1 Similarity Join Without Index Support

When processing a join operation with an arbitrary join predicate, the baseline approach is to consider the nested loop join (NLJ), involving two nested loops. Each of these two loops enumerates all points of the data set, producing by that the pairs of points, the distance between which is then computed and compared to threshold ϵ. In Figure 5.4 we show the pseudocode of the typical sequential version of NLJ running on CPU.

The NLJ algorithm above can be easily adapted to run in parallel on GPU, for example as follows. We create a separate thread for each iteration of the outer loop. The remained inner loop we put in the kernel function, as well as the distance computation and the comparison to the threshold.

```
algorithm sequentialNLJ(data set $\mathcal{D}$)
  for each $q \in \mathcal{D}$ do        // outer loop
    for each $x \in \mathcal{D}$ do      // inner loop: search all points $x$ which are similar to $q$
      if dist$(x, q) \leq \epsilon$ then
        report $(x, q)$ as a result pair or do some further processing on $(x, q)$
end
```

Figure 5.4: Sequential Algorithm for the Nested Loop Join.

During the complete run of the kernel function, which corresponds to the inner loop, the value in the outer loop (the current point) is fixed. This value is called the *query point* q of the thread, since the thread works like a similarity query, finding all those points from database, distance from q to which does not exceed ϵ. To optimize the access time, the query point q is always kept in a register of the processor.

Using GPU, it is possible to realize a truly parallel execution of a number m of incarnations of the outer loop, where m is the total number of ALUs of all multiprocessors, i.e. the number of multiprocessors times 32 (the warp size). The processing of the different warps happens in a quasi-parallel manner, allowing to operate on another warp of threads (which is ready-to-run) while some warp is blocked due to the latency delay of a DM access of one of its threads.

As we already told, threads are grouped (after being aggregating into warps) into thread groups, using the same the shared memory. Concretely in our case, the SM is responsible for storing the current point x of the inner loop for each thread group. Thereby, the current point x is copied by a kernel function from the DM into the SM, calculating after that the distance from x to the query point q. The threads, belonging to the same warp, are copying the same point from DM to SM, thus running perfectly simultaneously. The copy operation is to be done only once, however under the restriction that all threads of the warp have to wait until this relatively expensive operation is done. As a thread group typically consists of multiple warps, we need to make sure that the copy operation is performed only once per thread group. The latter can be done by synchronizing the threads of the thread group before and after the copy operation by using the API function `synchronizeThreadGroup()`. This API call yields blocking all threads of the thread group until the same point of execution is reached by all other threads in this TG (also from other warps). In the Figure 5.5 we present the pseudocode for this algorithm.

```
algorithm GPUsimpleNLJ(data set $\mathcal{D}$)        // host program executed on CPU
  deviceMem float $\mathcal{D}'[][] := \mathcal{D}[][]$;         // allocate memory in DM for the data set $\mathcal{D}$
  #threads := $n$;                       // number of points in $\mathcal{D}$
  #threadsPerGroup := 64;
  startThreads (simpleNLJKernel, #threads, #threadsPerGroup);      // one thread per point
  waitForThreadsToFinish();
end.

kernel simpleNLJKernel (int threadID)
  register float $q[] := \mathcal{D}'$[threadID][];       // copy the point from DM into the register
                                        // and use it as query point $q$
                                        // index is determined by the threadID
  for $i := 0 \dots n-1$ do                   // this used to be the inner loop in Figure 5.4
    synchronizeThreadGroup();
    shared float $x[] := \mathcal{D}'[i][]$;              // copy the current point $x$ from DM to SM
    synchronizeThreadGroup();              // Now all threads of the thread group can work with $x$
    if dist$(x, q) \leq \epsilon$ then
      report $(x, q)$ as a result pair using synchronized writing
      or do some further processing on $(x, q)$ directly in kernel
end.
```

Figure 5.5: Parallel Algorithm for the Nested Loop Join on the GPU.

It can happen, that all the data does not fit into the DM. In this case, a simple partitioning approach can be applied. When partitioning, it is important to ensure, that potential participators of join fall into the same partition. As consequence, we have overlapping partitions of size $2 \cdot \epsilon$, which means that some objects may be present in many partitions.

5.4.2 An Indexed Parallel Similarity Join Algorithm on GPU

The performance of the NLJ can significantly be improved when having an index structure as proposed in Section 5.3. In the case of architectures with sequential processing, the outer loop in the index version of NLJ is the same, whereas the inner loop is exchanged by an index-based search looking for candidates suitable as join partners of the current object of the outer loop. To find such candidates and refine them is usually essentially (orders of magnitude) cheaper in terms of related costs compared to the non-indexed version of NLJ. When adapting the indexed NLJ for parallel architecture of the GPU, we use the same idea as in the last section, namely to create a separate thread for each point of the outer loop. If we collect the points, the distance between which is small, to the same warp and TG, this has a positive influence on the performance, since similar paths in the index structure for those points are of importance.

After constructing an index, we can profit from two things at once. Firstly, we have a directory with such organization of points that facilitates the search. And secondly, we have additionally a clusterization of points in form of an array. The latter means that the points with neighboring addresses in the array are also supposed to be close to each other in the data space, at least when projecting on the first few dimensions. Our join algorithm, using the two effects, is shown in Figure 5.6

```
algorithm GPUindexedJoin(data set $\mathcal{D}$)
  deviceMem index idx := makeIndexAndSortData($\mathcal{D}$); // changes ordering of data points
  int #threads := $|\mathcal{D}|$, #threadsPerGroup := 64;
  for $i = 1$ ... (#threads/#threadsPerGroup) do
      // find the boundaries of the data page
    deviceMem float blockbounds[$i$][] := calcBlockBounds($\mathcal{D}$, blockindex);
  deviceMem float $\mathcal{D}'$[][] := $\mathcal{D}$[][];
  startThreads (indexedJoinKernel, #threads, #threadsPerGroup); // one thread per data point
  waitForThreadsToFinish ();
end.

algorithm indexedJoinKernel (int threadID, int blockID)
  register float $q$[] := $\mathcal{D}'$[threadID][];      // copy the point from DM into the register
  shared float myblockbounds[] := blockbounds[blockID][];
  for $x_i$ := 0 ... indexsize.$x$ do
    if IndexPageIntersectsBoundsDim1(idx,myblockbounds,$x_i$) then
      for $y_i$ := 0 ... indexsize.$y$ do
        if IndexPageIntersectsBoundsDim2(idx,myblockbounds,$x_i, y_i$) then
          for $z_i$ := 0 ... indexsize.$z$ do
            if IndexPageIntersectsBoundsDim3(idx,myblockbounds,$x_i, y_i, z_i$) then
              for $w$ := 0 ... IndexPageSize do
                synchronizeThreadGroup();
                shared float $p$[] :=GetPointFromIndexPage(idx,$\mathcal{D}', x_i, y_i, z_i, w$);
                synchronizeThreadGroup();
                if dist($p, q$) $\le \epsilon$ then
                  report ($p, q$) as a result pair using synchronized writing
end.
```

Figure 5.6: Algorithm for Similarity Join on GPU with Index Support.

Unlike a sequential indexed version of NLJ, where we perform outer loop, in our algorithm we create many threads at once, one thread in place of each iteration of the outer loop (in other words, one thread per query point q). Since the points in the array are clustered, the respective query points are close to each other, and the join partners of all query points in a thread group are likely to reside in the same branches of the index as well. Our kernel method performs now iteration over three loops (one loop per index level) and determines for each partition if the point is inside the partition or, at least its distance to the boundary of the partition does not exceed ϵ. The corresponding subnode is accessed if the corresponding partition is able to contain join partners of the current point of the thread. When considering the warps which operate fully synchronously, a node is accessed if at least one of the query points of the warps is close enough to (or inside) the respective partition.

When dealing with both indexed and non-indexed nested loop join on GPU, the important question is how the pairs we get as result are processed. For example, to provide density-based clustering (cf. Section 5.5), it is usually sufficient to return a counter with the number of join partners. Reporting the pairs themselves can also be easily done via copying of DM buffer to the CPU after terminating of all kernel threads. To avoid that many threads simultaneously write to the same buffer, it must be ensured that the resulting pairs are written into this buffer synchronously. The CUDA API is a good assistant in this case. Certain atomic operations (such as atomic increment of a buffer pointer), provided by CUDA API ensure this kind of synchronized writing. Our similarity join methods also provide buffer overflow protection. If the buffer gets full, all threads terminate and continue only after the buffer is emptied by the CPU.

5.5 Similarity Join to Support Density-based Clustering

As we already mentioned in Section 5.4, the similarity join represents an important building block allowing to facilitate a wide spectrum of Data Mining tasks such as classification [153], clustering [46], [64], association rule mining [104] and outlier detection [28]. In this section, we show how the GPU-based similarity join can be applied for effective support of the density-based clustering algorithm DBSCAN [46].

5.5.1 Basic Definitions and Sequential DBSCAN

As a basis of density-based clustering, the idea is taken that clusters are supposed to be areas of high point density, separated by areas of significantly lower point density. We can formalize the notion of point density using two parameters, called $\epsilon \in \mathbb{R}^+$ and $MinPts \in \mathbb{N}^+$. The central concept is the *core object* of a cluster. The latter is defined as a data object x, for which at least $MinPts$ objects (including x itself) are in its ϵ-neighborhood $N_\epsilon(x)$, which corresponds to a sphere of radius ϵ. Formally:

Definition 4 *(Core Object)*
Let $\mathcal{D}$ be a set of n objects from $\mathbb{R}^d$, $\epsilon \in \mathbb{R}^+$ and $MinPts \in \mathbb{N}^+$. An object $x \in \mathcal{D}$ is a core object, if and only if

$$|N_\epsilon(x)| \geq MinPts, \text{ where } N_\epsilon(x) = \{x' \in \mathcal{D} : ||x' - x|| \leq \epsilon\}.$$

It should be noted, that this definition is related to Definition 1. Belonging of two points to the same cluster is characterized in density-based clustering by the notions *direct density reachability, and density connectedness*. Direct density reachability is formally defined as follows:

Definition 5 *(Direct Density Reachability)*
Let $x, x' \in \mathcal{D}$. x' is called directly density reachable from x (in symbols: $x \lhd x'$) if and only if

1. *x is a core object in $\mathcal{D}$, and*
2. *$x' \in N_\epsilon(x)$.*

If both x and x' are core objects, then $x \lhd x'$ is equivalent to $x \rhd x'$. The density connectedness is the transitive and represents a symmetric closure of the direct density reachability:

Definition 6 *(Density Connectedness)*
Two objects x and x' are called density connected (in symbols: $x \bowtie x'$) if and only if there is a sequence of core objects $(x_1, ..., x_m)$ of arbitrary length m such that

$$x \rhd x_1 \rhd ... \lhd x_m \lhd x'.$$

It is to be noted, that the reachability direction in the definition above 6 is essential only for the first object x and the last object x'. All directions in-between are interchangable for core objects due to the note after Definition 5.

Finally, we can give in case of density-based clustering a formal definition of a cluster as a maximal set of density connected objects:

Definition 7 *(Density-based Clustering)*
A disjunct set $C_1 \subseteq \mathcal{D}, \ldots, C_N \subseteq \mathcal{D}$ is called a clustering of $\mathcal{D}$ (each C_i is correspondingly called a cluster) if and only if the following two conditions hold:

1. *Density connectedness: $\forall i = \overline{1, N}, \forall x, x' \in C_i : x \bowtie x'$.*
2. *Maximality: $\forall i = \overline{1, N}, \forall x \in C_i, \forall x' \in \mathcal{D} \setminus \bigcup_{j=1}^{N} C_j : \neg(x \bowtie x')$.*

An implementation of the cluster notion according to Definition 7 represents the algorithm DBSCAN [46]. This algorithm uses a data structure called *seed list* S consisting of seed objects for cluster expansion. More precisely, the algorithm proceeds as follows:

1. Mark all objects as *unprocessed*.
2. Consider an arbitrary unprocessed object $x \in \mathcal{D}$.
3. If x is a core object, assign a new cluster ID C, and do step (4) for all elements $x' \in N_\epsilon(x)$ which do not yet have a cluster ID:
4. (a) mark the element x' with the cluster ID C and
   (b) insert the object x' into the seed list S.
5. While S is not empty repeat step 6 for all elements $s \in S$:
6. If s is a core object, do step (7) for all elements $x' \in N_\epsilon(s)$ which do not yet have any cluster ID:
7. (a) mark the element x' with the cluster ID C and
   (b) insert the object x' into the seed list S.
8. If there are still unprocessed objects in the database, continue with step (2).

A snapshot of DBSCAN during cluster expansion, illustrating the algorithmic paradigm, is shown in Figure 5.7. The light grey cluster on the left side depicts a cluster that has been already processed. The algorithm currently proceeds with expansion of the dark grey cluster we can see on the right side. The seed list S at this moment contains one object, namely the object x. As it can be counted more than $MinPts = 3$ objects in ϵ-neighborhood of x ($|N_\epsilon(x)| = 6$, including x itself), it can be concluded that x is a core object. Two of these objects, x' and x'' have not been processed yet and are therefore inserted into S. Continuing this way, the cluster is iteratively expanded until the seed list becomes empty. After that, the algorithm proceeds with an arbitrary unprocessed object until all objects have been finally processed.

Speaking about the complexity of the algorithm, it can be noted that every object of the database is considered only once either in Step 2 *or* in Step 6 (exclusively), which yields a complexity equal to n times the complexity of $N_\epsilon(x)$. The latter is linear (in n) if there is no index structure, and sublinear or even $O(\log(n))$ in the presence of a multidimensional index structure. The resulting clustering is deterministic for core objects. For objects, which could belong to many clusters, the belonging to a certain cluster is defined by the order of object processing.

5.5.2 GPU-supported DBSCAN

To effectively speed up DBSCAN via supporting it by GPU, we need at first to recognize the algorithm phases requiring most of the processing time. These are the following two stages:

1. Determination of the core object property.

2. Cluster expansion by computing the transitive closure of the direct density reachability relation.

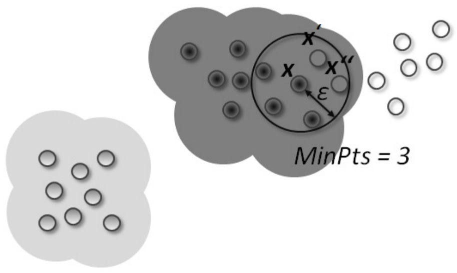


Figure 5.7: Sequential Density-based Clustering.

```
algorithm GPUdbscanNLJ(data set $\mathcal{D}$)        // host program executed on CPU
  deviceMem float $\mathcal{D}'[][] := \mathcal{D}[][]$;           // allocate memory in DM for the data set $\mathcal{D}$
  deviceMem int counter [$n$];                  // allocate memory in DM for counter
  #threads := $n$;                            // number of points in $\mathcal{D}$
  #threadsPerGroup := 64;
  startThreads (GPUdbscanKernel, #threads, #threadsPerGroup);      // one thread per point
  waitForThreadsToFinish();
  copy counter from DM to main memory ;
end.

kernel GPUdbscanKernel (int threadID)
  register float $q[] := \mathcal{D}'$[threadID][];       // copy the point from DM into the register
                                       // and use it as query point $q$
                                       // index is determined by the threadID
  for $i := 0$ ... threadID do                 // option 1 OR
  for $i := 0$ ... $n-1$ do                   // option 2
    synchronizeThreadGroup();
    shared float $x[] := \mathcal{D}'[i][]$;               // copy the current point $x$ from DM to SM
    synchronizeThreadGroup();             // Now all threads of the thread group can work with $x$
    if dist$(x, q) \le \epsilon$ then
      atomicInc (counter[$i$]); atomicInc (counter[threadID]); // option 1 OR
      inc counter[threadID];                          // option 2
end.
```

Figure 5.8: Parallel Algorithm for the Nested Loop Join to Support DBSCAN on GPU.

The performance of the first stage can be effectively improved by means of GPU-based similarity join. For checking the core object property, for each point we need to count the number of objects falling into its ϵ-neighborhood. Basically, this can be implemented by a self join. However, we need to adjust the algorithm for self-join described in Section 5.4 to make it suitable to support this task. The classical self-join only counts the total number of pairs of data objects, the distance between which is not more than a given threshold ϵ. To test an object if it a core one, we need a self-join with a counter assigned to each object. Each time when detecting a new pair satisfying the join condition, the counter of both objects is to be incremented.

We propose here two different variants how to implement the self-join to support DBSCAN on GPU. The two versions are shown in Figure 5.8 in form of a pseudocode. Adjustments over the basic algorithm for nested loop join (cf. Figure 5.5) are given in darker color. Like in case of the simple algorithm for nested loop join, for each point q of the outer loop a dedicated thread with a unique *threadID* is created. In both variants of the self-join for DBSCAN proposed here, we use an array *counter* to store the number of neighbors for each object. When the join condition on a pair of objects (x, q)

is fulfilled, we have a choice of two options how to increment the counters of the objects. Option 1 is at first to add the counter of x and then the counter of q using the atomic operation `atomicInc()` (cf. Section 4.3). This yields the synchronization of all participating threads. As already mentioned before, the atomic operations serve to assure the correctness of the result, when various threads simultaneously try to increment the counters of objects.

In clustering, we typically have many core objects which causes a large number of synchronized operations limiting the parallelism. Therefore, we also implemented option 2 which ensures correctness without synchronized operations. As soon as a pair of objects (x, q) satisfies the join condition, we only increment the counter of point q. Point q corresponds to the point of the outer loop for which the separate thread has been created, which means q is exclusively associated with the $threadID$. Therefore, the cell $counter[threadID]$ can be safely incremented with the ordinary, non-synchronized operation `inc()`. Since no other point is associated with the same $threadID$ as q no collision can happen. However, note that unlike option 1, for each point of the outer loop, the inner loop needs to consider all other points. Otherwise results are missed. Recall that for the conventional sequential nested loop join (cf. Figure 5.4) it suffices to consider in the inner loop only those points which have not been processed so far. Already processed points can be excluded because if they are join partners of the current point, this has already been detected. The same holds for option 1. Because of parallelism, we can not state which objects have been already processed. However, it is still sufficient when each object searches in the inner loop for join partners among those objects which would appear later in the sequential processing order. This is because all other object are addressed by different threads. Option 2 requires checking all objects since only one counter is incremented. With sequential processing, option 2 would thus duplicate the workload. However, as our results in Section 5.6 demonstrate, option 2 can pay-off under certain conditions since parallelism is not limited by synchronization.

After detecting the core objects, we can begin to expand clusters starting from these core objects. GPU can effectively support as well this second stage of DBSCAN. For cluster expansion, it is required to calculate the transitive closure of the direct density reachability relation. Recall that this is closely connected to the core object property since all objects within the ϵ neighborhood of a core object x are directly density reachable from x. The calculation of the transitive closure can be done using standard algorithms, the most well-known among which is the algorithm of Floyd-Warshall. A highly parallel version of the Floyd-Warshall algorithm on GPU has been recently proposed [96], but this is beyond the scope of this paper.

5.6 Experimental Evaluation

To evaluate the performance of Data Mining on the GPU, we performed various experiments on synthetic data sets. The implementation for all variants is written in C and all experiments are performed on a workstation with Intel Core 2 Duo CPU E4500 2.2 GHz and 2 GB RAM which is supplied with a Gainward NVIDIA GeForce GTX280 GPU (240 SIMD-processors) with 1GB GDDR3 SDRAM.

5.6.1 Evaluation of Similarity Join on the GPU

The performance of similarity join on the GPU, is validated by the comparison of four different variants for executing similarity join:

1. Nested loop join (NLJ) on the CPU
2. NLJ on the CPU with index support (as described in Section 5.3)
3. NLJ on the GPU
4. NLJ on the GPU with index support (as described in Section 5.3)

For each version we determine the speedup factor by the ratio of CPU runtime and GPU runtime. For this purpose we generated three 8-dimensional synthetic data sets of various sizes (up to 10 million (m) points) with different data distributions, as summarized in Table 5.1. Data set DS_1 contains uniformly distributed data. DS_2 consists of five Gaussian clusters which are randomly distributed in feature space (see Figure 5.9(a)). Similar to DS_2, DS_3 is also composed of five Gaussian clusters, but the clusters are correlated. An illustration of data set DS_3 is given in Figure 5.9(b). The threshold ϵ was selected to obtain a join result where each point was combined with one or two join partners on average.

Evaluation of the Size of the Data Sets. Figure 5.10 displays the runtime in seconds and the corresponding speedup factors of NLJ on the

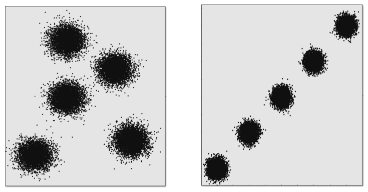

(a) Random Clusters. (b) Linear Clusters.

Figure 5.9: Illustration of the data sets DS_2 and DS_3.

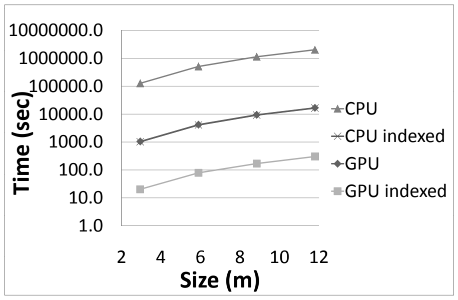
Name	Size	Distribution
DS_1	3m - 10m points	uniform distribution
DS_2	250k - 1m points	normal distribution, gaussian clusters
DS_3	250k - 1m points	normal distribution, gaussian clusters

Table 5.1: Data Sets for the Evaluation of the Similarity Join on the GPU.

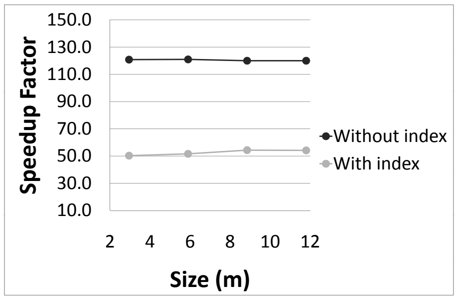
CPU with/without index support and NLJ on the GPU with/without index support in logarithmic scale for all three data sets DS_1, DS_2 and DS_3. The time needed for data transfer from CPU to the GPU and back as well as the (negligible) index construction time has been included. The tests on data set DS_1 were performed with a join selectivity of $\epsilon = 0.125$, and $\epsilon = 0.588$ on DS_2 and DS_3 respectively.

NLJ on the GPU with index support performs best in all experiments, independent of the data distribution or size of the data set. Note that, due to massive parallelization, NLJ on the GPU without index support outperforms CPU without index by a large factor (e.g. 120 on 1m points of normal distributed data with gaussian clusters). The GPU algorithm with index support outperforms the corresponding CPU algorithm (with index) by a factor of 25 on data set DS_2. Remark that for example the overall improvement of the indexed GPU algorithm on data set DS_2 over the non-indexed CPU version is more than 6,000. This results demonstrate the potential of boosting performance of database operations with designing specialized index structures and algorithms for the GPU.

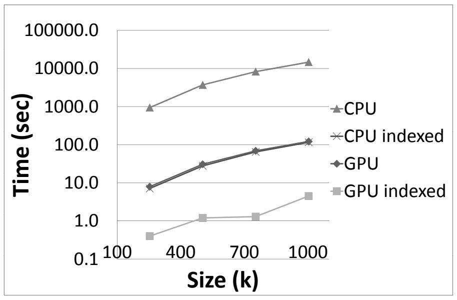
Evaluation of the Join Selectivity. In these experiments we test the impact of the parameter ϵ on the performance of NLJ on GPU with index support and use the indexed implementation of NLJ on the CPU as benchmark. All experiments are performed on data set DS_2 with a fixed size of 500k data points. The parameter ϵ is evaluated in a range from 0.125 to 0.333.



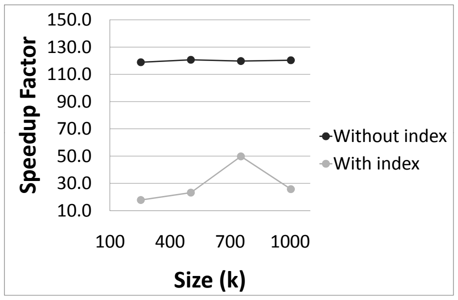
(a) Runtime on Data Set DS_1.



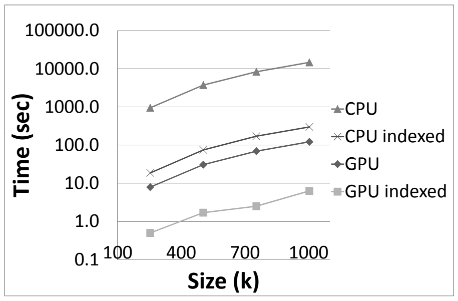
(b) Speedup on Data Set DS_1.



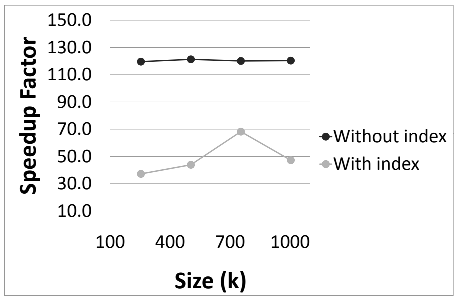
(c) Runtime on Data Set DS_2.



(d) Speedup on Data Set DS_2.

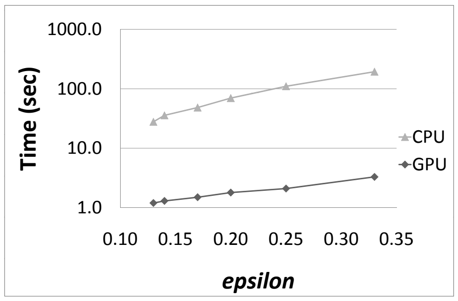


(e) Runtime on Data Set DS_3.

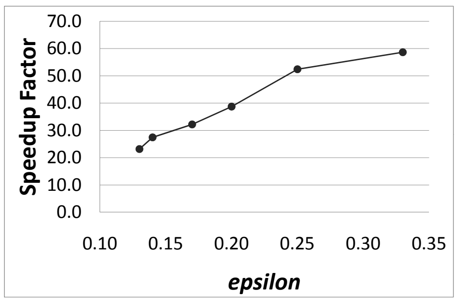


(f) Speedup on Data Set DS_3.

Figure 5.10: Evaluation of the NLJ on CPU and GPU with and without Index Support w.r.t. the Size of Different Data Sets.



(a) Runtime on Data Set DS_2.



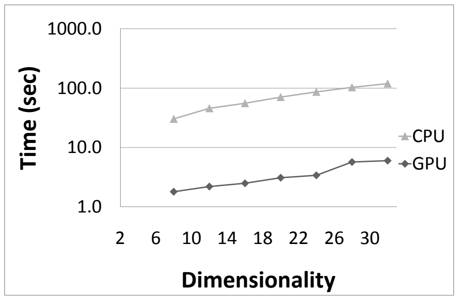
(b) Speedup on Data Set DS_2.

Figure 5.11: Impact of the Join Selectivity on the NLJ on GPU with Index Support.

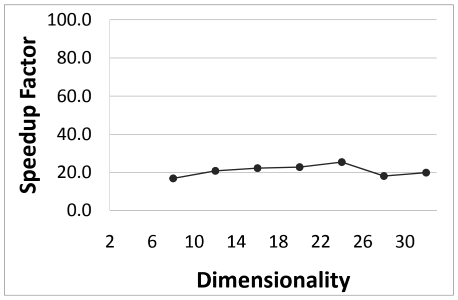
Figure 5.11(a) shows that the runtime of NLJ on GPU with index support increases for larger ϵ values. However, the GPU version outperforms the CPU implementation by a large factor (cf. Figure 5.11(b)), that is proportional to the value of ϵ. In this evaluation the speedup ranges from 20 for a join selectivity of 0.125 to almost 60 for $\epsilon = 0.333$.

Evaluation of the Dimensionality. These experiments provide an evaluation with respect to the dimensionality of the data. As in the experiments for the evaluation of the join selectivity, we use again the indexed implementations both on CPU and GPU and perform all tests on data set DS_2 with a fixed number of 500k data objects. The dimensionality is evaluated in a range from 8 to 32. We also performed these experiments with two different settings for the join selectivity, namely $\epsilon = 0.588$ and $\epsilon = 1.429$.

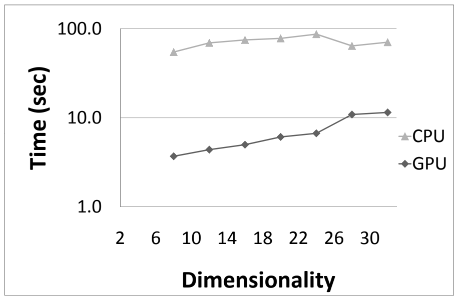
Figure 5.12 illustrates that NLJ on GPU outperforms the benchmark method on CPU by factors of about 20 for $\epsilon = 0.588$ to approximately 70 for $\epsilon = 1.429$. This order of magnitude is relatively independent of the data dimensionality. As in our implementation the dimensionality is already known at compile time, optimization techniques of the compiler have an impact on the performance of the CPU version as can be seen especially in Figure 5.12(c). The sharp bend of the curve in Figure 5.12(d) is explained by the same. However the dimensionality also affects the implementation on GPU, because higher dimensional data come along with a higher demand of shared memory. This overhead affects the number of threads that can be executed in parallel on the GPU.



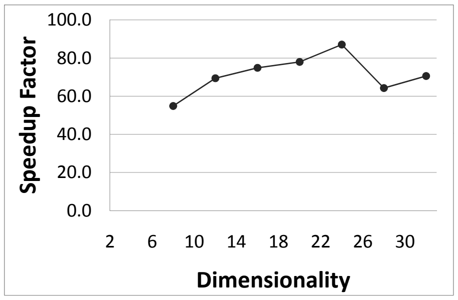
(a) Runtime on Data Set D_2 ($\epsilon = 0.588$).



(b) Speedup on Data Set D_2 ($\epsilon = 0.588$).



(c) Speedup on Data Set D_2 ($\epsilon = 1.429$).



(d) Speedup on Data Set D_2 ($\epsilon = 1.429$).

Figure 5.12: Impact of the Dimensionality on the NLJ on GPU with Index Support.

5.6.2 Evaluation of GPU-supported DBSCAN

As described in Section 5.5.2, we suggest two different variants to implement the self-join to support DBSCAN on GPU, whose characteristics are briefly reviewed in the following:

1. Increment of the counters regarding a pair of objects (x, q) that fulfills the join condition is done by the use of an atomic operation that involves *synchronization* of all threads.
2. Increment of the counters can be performed *without synchronization* but with duplicated workload instead.

We evaluate both options on a synthetic data set with 500k points generated as specified as DS_1 in Table 5.1. Figure 5.13 displays the runtime of both options. For $\epsilon \leq 0.6$, the runtime is in the same order of magnitude, the synchronized variant 1 being slightly more efficient. From this point on, the non-synchronized variant 2 is clearly outperforming variant 1 since parallelism is not limited by synchronization. When the number of similar objects is big enough ($\epsilon \approx 0.6$), an additional synchronization is too expensive.

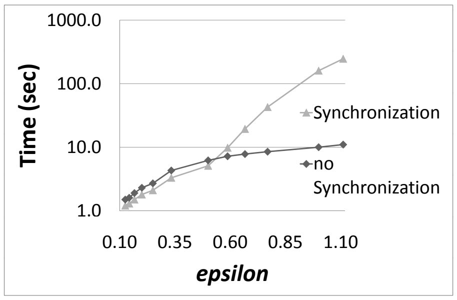


Figure 5.13: Evaluation of two versions for the self-join on GPU w.r.t. the join selectivity.

5.7 Conclusions

In this chapter, we demonstrated how Graphics processing Units (GPU) can effectively support highly complex Data Mining tasks. In particular, we focused on clustering. With the aim of finding a natural grouping of an unknown data set, clustering certainly is among the most widely spread Data Mining tasks with countless applications in various domains. We selected a well-known clustering algorithm, the density-based algorithm DBSCAN and proposed a new algorithm illustrating how to effectively support clustering on GPU. Our proposed algorithms are customized to the special environment of the GPU which is most importantly characterized by extreme parallelism at low cost. A single GPU consists of a large number of processors. As building blocks for effective support of DBSCAN, we proposed a parallel version of the similarity join and an index structure for efficient similarity search. Going beyond the primary scope of this chapter, these building blocks are applicable to support a wide range of Data Mining tasks, including outlier detection, association rule mining and classification. Our extensive experimental evaluation emphasizes the potential of the GPU for high-performance Data Mining. In our ongoing work, we develop further algorithms to support more specialized Data Mining tasks on GPU.

Chapter 6

K-means Clustering using GPU

In this chapter we continue applying GPU-based computing and introduce a highly parallel algorithm for k-means-based clustering specially designed for execution in GPU environment. By means of an extensive experimental evaluation, we demonstrate the superiority of our algorithm running on GPU compared to their competitors running on CPU.
Parts of the material presented in this chapter have been published in Paper [25]. In this paper, Andrew Zherdin has codesigned the main concepts for parallelization as well as implemented and carried out a part of experiments. Robert Noll has implemented and optimized the most of source code. His task was also the choice and installation of the used hardware. Bianca Wackersreuther has taken an important part in the development and optimization of the algorithms and experiments. Christian Böhm and Claudia Plant have supervised this work and essentially improved its results.

6.1 Introduction

In this chapter, we concentrate on application possibilities of using the computational power of GPUs for such important *Data Mining* task as clustering.

As already noticed in the introduction of the previous Chapter, the aim of clustering is to find a natural grouping of a data set. To be able to do this, a specification of the notion of similarity among objects is needed. Thereby, objects not fitting well to any of the clusters are counted as outliers and should be left unassigned. A reader can refer to introduction of the previous Chapter, where we have already considered in detail the notion of clustering.

To demonstrate that also highly complex Data Mining tasks can be efficiently implemented using novel parallel algorithms, we propose a parallel version of K-means clustering [122] algorithm, which follows an algorithmic paradigm that is very different from density-based clustering, considered in the previous chapter.

When implementing algorithms for GPU, we used NVIDIA's technology *Compute Unified Device Architecture* (CUDA) [2].

The remainder of this chapter is organized as follows: Section 6.2 represents a general review of the related work in GPU processing with particular focus on clustering. Section 6.3 is dedicated to GPU-capable algorithms for density-based and partitioning clustering. Section 6.4 presents the results of an extensive experimental evaluation of our approach, and Section 6.5 summarizes the chapter and suggests directions for future research.

6.2 Related Work

In this section, we provide a review of the related work in GPU processing with particular focus on clustering.

Considering recent research in the field of Data Mining with focus on clustering, we should to notice two papers, that pass on the use of CUDA. Using NVIDIA GeForce 6800 GT, the authors of [36] introduce a clustering approach that extends the basic idea of K-means via simultaneous computation on GPU of all the distances from a single input centroid to all objects at one time. By that, the authors succeeded to demonstrate the capability of GPUs of the high computational power and pipeline, especially for core operations, like comparisons and distance calculations. A yet another approach that is designed to execute clustering on data streams exemplifies a wide practical field of GPU-based clustering.

In paper [172], the authors exploit the multi-pass rendering and multi shader program constants to parallelize the K-means algorithm. Applying this realization on NVIDIA 5900 and NVIDIA 8500 graphics processors, the authors obtain significant performance gain for both various data sizes and cluster sizes.

Despite of good results, the algorithms of the two papers are GPU-specific and unfortunately not portable to different GPU models, like CUDA-approaches are.

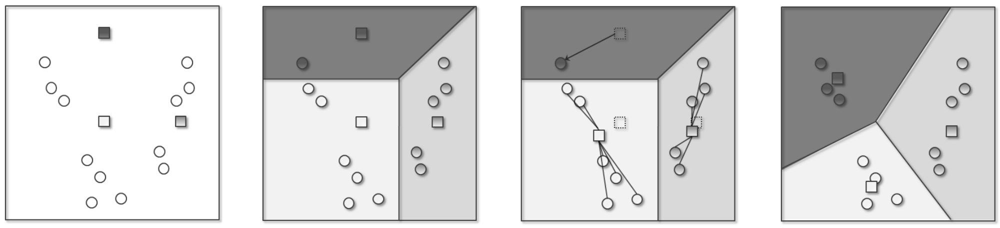
6.3 K-means Clustering on GPU

6.3.1 The Algorithm K-means

A well-established partitioning clustering method is the K-means clustering algorithm [122]. K-means requires a metric distance function in vector space. In addition, the user has to specify the number of desired clusters k as an input parameter. Usually K-means starts with a certain partitioning of the objects into k clusters. After this initialization, the algorithm iteratively performs the following two steps until convergence: (1) Update centers: For each cluster, compute the mean vector of its assigned objects. (2). Re-assign objects: Assign each object to its closest center. The algorithm converges as soon as no object changes its cluster assignment during two subsequent iterations.

Figure 6.1 illustrates an example run of K-means for $k = 3$ clusters. Figure 6.1(a) shows the situation after random initialization. In the next step, every data point is associated with the closest cluster center (cf. Figure 6.1(b)). The resulting partitions represent the Voronoi cells generated by the centers. In the following step of the algorithm, the center of each of the k clusters is updated, as shown in Figure 6.1(c). Finally, assignment and update steps are repeated until convergence.

In most cases, fast convergence can be observed. The optimization function of K-means is well defined. The algorithm minimizes the sum of squared distances of the objects to their cluster centers. However, K-means is only guaranteed to converge towards a local minimum of the objective function. The quality of the result strongly depends on the initialization. Finding that clustering with k clusters minimizing the objective function actually is a NP-hard problem, for details see e.g. [130]. In practice, it is therefore recommended to run the algorithm several times with different random initializations and keep the best result. For large data sets, however, often only a very limited number of trials is feasible. Parallelizing K-means in GPU allows for a more comprehensive exploration of the search space of all potential clusterings and thus provides the potential to obtain a good and reliable clustering even for very large data sets.



(a) Initialization. (b) Assignment. (c) Recalculation. (d) Termination.

Figure 6.1: Sequential Partitioning Clustering by the K-means Algorithm.

6.3.2 CUDA-K-means

In K-means, most computing power is spent in step (2) of the algorithm, i.e. re-assignment which involves distance computation and comparison. The number of distance computations and comparisons in K-means is $O(k \cdot i \cdot n)$, where i denotes the number of iterations and n is the number of data points.

The CUDA-K-meansKernel. In K-means clustering, the cluster assignment of each data point is determined by comparing the distances between that point and each cluster center. This work is performed in parallel by the CUDA-K-meansKernel. The idea is, instead of (sequentially) performing cluster assignment of one single data point, we start many different cluster assignments at the same time for different data points. In detail, one single thread per data point is generated, all executing the CUDA-K-meansKernel. Every thread which is generated from the CUDA-K-meansKernel (cf. Figure 6.2) starts with the ID of a data point x which is going to be processed. Its main tasks are, to determine the distance to the next center and the ID of the corresponding cluster.

A thread starts by reading the coordinates of the data point x into the register. The distance of x to its closest center is initialized by ∞ and the assigned cluster is therefore set to `null`. Then a loop encounters all $c_1, c_2, \ldots, c_k$ centers and considers them as potential clusters for x. This is done by all threads in the thread group allowing a maximum degree of intra-group parallelism. Finally, the cluster whose center has the minimum distance to the data point x is reported together with the corresponding distance value using synchronized writing.

The Main Program for CPU. Apart from initialization and data transfer from main memory (MM) to DM, the main program consists of a loop starting the CUDA-K-meansKernel on the GPU until the clustering converges. After the parallel operations are completed by all threads of the group, the following steps are executed in each cycle of the loop:

1. Copy distance of processed point x to the nearest center from DM into MM.
2. Copy cluster, x is assigned to, from DM into MM.
3. Update centers.
4. Copy updated centers to DM.

A pseudocode of these procedures is illustrated in Figure 6.2.

6.4 Evaluation of CUDA-K-means

To analyze the efficiency of K-means clustering on the GPU, we present experiments with respect to different data set sizes, number of clusters and dimensionality of the data. As benchmark we apply a single-threaded implementation of K-means on the CPU to determine the speedup of the implementation of K-means on the GPU. As the number of iterations may vary in each run of the experiments, all results are normalized by a number of 50 iterations both on the GPU and the CPU implementation of K-means. All experiments are performed on synthetic data sets as described in detail in each of the following settings.

Evaluation of the Size of the Data Set. For these experiments we created 8-dimensional synthetic data sets of different size, ranging from 32k to 2m data points. The data sets consist of different numbers of random clusters, generated as as specified as DS_1 in Table 5.1.

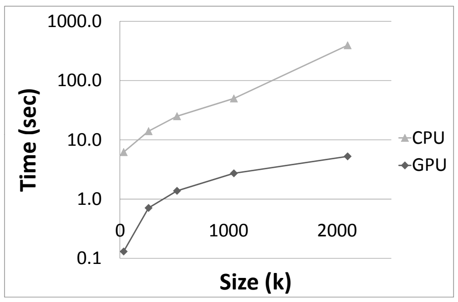
Figure 6.3 displays the runtime in seconds in logarithmic scale and the corresponding speedup factors of CUDA-K-means and the benchmark method on the CPU for different numbers of clusters. The experiments were carried out multiple times with random initializations. The time needed for data transfer from CPU to GPU and back has been included. The corresponding speedup factors are given in Figure 6.3(d). Once again, these

```
algorithm CUDA-K-means(data set $\mathcal{D}$, int $k$)                 // host program executed on CPU
  deviceMem float $\mathcal{D}'[][] := \mathcal{D}[][]$;                  // allocate memory in DM for the data set $\mathcal{D}$
  #threads := $|\mathcal{D}|$;                                            // number of points in $\mathcal{D}$
  #threadsPerGroup := 64;
  deviceMem float $Centroids[][]$ := initCentroids();    // allocate memory in DM for the
                                                        // initial centroids
  double $actCosts := \infty$;                            // initial costs of the clustering

  repeat
    $prevCost := actCost$;
    startThreads (CUDA-K-meansKernel, #threads, #threadsPerGroup);    // one thread per point
    waitForThreadsToFinish();
    float $minDist := minDistances[threadID]$;           // copy the distance to the nearest
                                                        // centroid from DM into MM
    float $cluster := clusters[threadID]$;               // copy the assigned cluster from DM into MM
    double $actCosts$ := calculateCosts();               // update costs of the clustering
    deviceMem float $Centroids[][]$ := calculateCentroids(); // copy updated centroids to DM
  until $|actCost - prevCost| < threshold$               // convergence
end.


kernel CUDA-K-meansKernel (int $threadID$)
  register float $x[] := \mathcal{D}'[threadID][]$;    // copy the point from DM into the register
  float $minDist := \infty$;                      // distance of $x$ to the next centroid
  int $cluster$ := null;                          // ID of the next centroid (cluster)
  for $i := 1 \ldots k$ do                         // process each cluster
    register float $c[] := Centroids[i][]$      // copy the actual centroid from DM into the register
    double $dist$ := distance($x$,$c$);
    if $dist < minDist$ then
      $minDist := dist$;
      $cluster := i$;
  report($minDist$, $cluster$);                   // report assigned cluster and distance using synchronized writing
end.
```

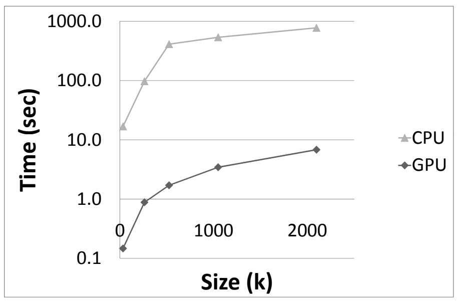
Figure 6.2: Parallel Algorithm for K-means on the GPU.

experiments support the evidence that the performance of Data Mining approaches on GPU outperform classic CPU versions by significant factors. Whereas a speedup of approximately 10 to 100 can be achieved for relatively small number of clusters, we obtain a speedup of about 1000 for 256 clusters, that is even increasing with the number of data objects.

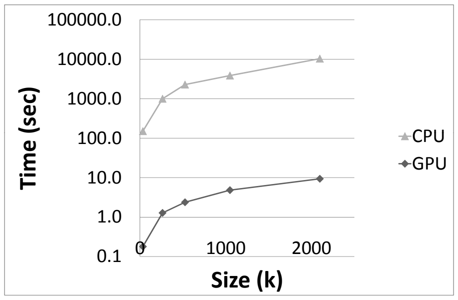
Evaluation of the Impact of the Number of Clusters. We performed several experiments to validate CUDA-K-means with respect to the number of clusters K. Figure 6.4 shows the runtime in seconds of CUDA-K-means compared with the implementation of K-means on the CPU on 8-dimensional synthetic data sets that contain different numbers of clusters,



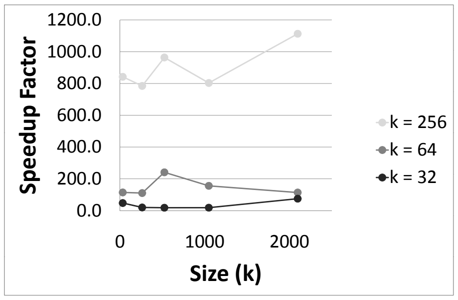
(a) Runtime for 32 clusters.



(b) Runtime for 64 clusters.



(c) Runtime for 256 clusters.



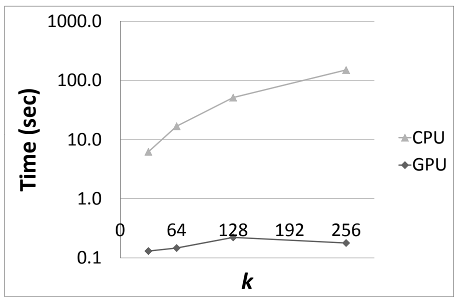
(d) Speedup for 32, 64 and 256 clusters.

Figure 6.3: Evaluation of CUDA-K-means w.r.t. the Size of the Data Set.

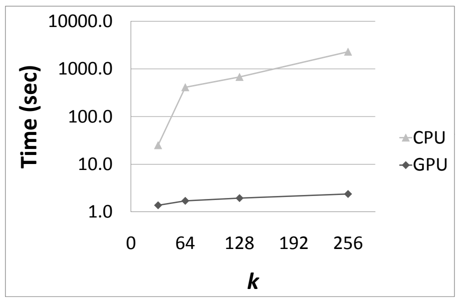
ranging from 32 to 256, again together with the corresponding speedup factors in Figure 6.4(d).

The experimental evaluation of K on a data set that consists of 32k points results in a maximum performance benefit of more than 800 compared to the benchmark implementation. For 2m points the speedup ranges from nearly 100 up to even more than 1,000 for a data set that comprises 256 clusters. In this case the calculation on the GPU takes approximately 5 seconds, compared to almost 3 hours on the CPU. Therefore, we determine that due to massive parallelization, CUDA-K-means outperforms CPU by large factors, that are even growing with K and the number of data objects n.

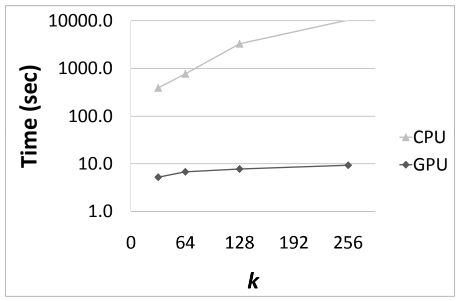
Evaluation of the Dimensionality. These experiments provide an evaluation with respect to the dimensionality of the data. We perform all tests on synthetic data consisting of 16k data objects. The dimensionality of the test data sets vary in a range from 4 to 256. Figure 6.5(b) illustrates that



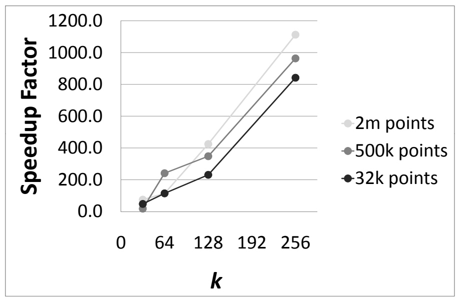
(a) Runtime for 32k points.



(b) Runtime for 500k points.



(c) Runtime for 2m points.



(d) Speedup for 32k, 500k and 2m points.

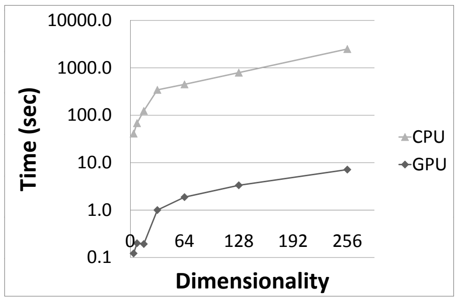
Figure 6.4: Evaluation of CUDA-K-means w.r.t. the number of clusters K.

CUDA-K-means outperforms the benchmark method K-means on the CPU by factors of 230 for 128-dimensional data to almost 500 for 8-dimensional data. On the GPU and the CPU, the dimensionality affects possible compiler optimization techniques, like loop unrolling as already shown in the experiments for the evaluation of the similarity join on the GPU.

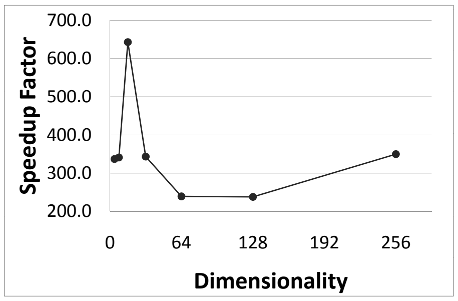
In summary, the results of this section demonstrate the high potential of boosting performance of complex Data Mining techniques by designing specialized index structures and algorithms for the GPU.

6.5 Conclusions

In this chapter, we demonstrated how Graphics processing Units (GPU) can effectively support such highly complex Data Mining task as clustering. With the aim of finding a natural grouping of an unknown data set, clustering certainly is among the most widely spread Data Mining tasks with countless ap-



(a) Runtime.



(b) Speedup.

Figure 6.5: Impact of the Dimensionality of the Data Set on CUDA-K-means.

plications in various domains. We selected the iterative algorithm K-means and proposed an algorithm illustrating how to effectively support clustering on GPU. Our proposed algorithms are customized to the special environment of the GPU which is most importantly characterized by extreme parallelism at low cost. To illustrate that not only local density-based clustering can be efficiently performed on GPU, we proposed in this chapter a parallelized version of K-means clustering. Our extensive experimental evaluation emphasizes the potential of the GPU for high-performance Data Mining. In our ongoing work, we develop further algorithms to support more specialized clustering tasks on GPU, including for example subspace and correlation clustering and medical image processing.

Part III

Models-based Data Mining

Chapter 7

Compact Model-Based Descriptions in Approximate Clustering of Time Series

Clustering time series is related to time-accuracy trade-off one has to confront with. Dealing with full-length time series trends to high runtime and storage costs on one hand, whereas compressing time series (for instance, via dimensionality reduction) and having to do with this compressed representation trends to low accuracy. An intelligent method of reducing the length of time series is therefore essential. We propose a mathematical model-based compression approach for time series which examines dependencies between different time series. The resulting time series represents a combination of a set of characteristic time series. Thereby we have only dependency upon the number of reference time series instead of the full length, which brings us considerable gain in runtime and storage costs. We show experimentally that even a not big number of reference time series can be used to achieve high accurate results at simultaneously sank runtime.
The basic ideas contained in this chapter have been published in Paper [106]. In this paper, Andrew Zherdin's part was developing the idea of the mathematical model for the approximation. Here, Andrew Zherdin has also shown the applicability of Mahalanobis-Distance. Besides that, Andrew Zherdin has carried out the experiments. The application of the approximation for clustering and needed experiments were proposed by Alexey Pryakhin, Matthias Renz and Peer Kröger. Hans-Peter Kriegel has supervised this work.

7.1 Introduction

Clustering time series data is a very significant task in Data Mining, appearing very often in a large number of diverse application areas including stock marketing, environmental analysis, astronomy, molecular biology and medical analysis. It is not a secret, that the length of time series in such application fields is usually enormous, which of course has a huge negative influence on the runtime of the clustering process. A lot of recent research work has focused therefore on efficient methods for similarity search in clustering of time series.

Time series are sequences of discrete quantitative data assigned to specific moments in time, i.e. a time series a is a sequence of values $a = \langle t_1, \ldots, t_m \rangle$, where t_i is the value at time slot i. This sequence is often also taken as a $m-$dimensional feature vector, i.e. $a \in \mathbb{R}^m$.

The performance of clustering algorithms for time series data is basically restricted by the cost needed for comparing pairs of time series (i.e. the processing cost of the used distance function). As already mentioned above, time series are usually very large counting several thousands of values per sequence. Hence, the comparison of two time series can be very expensive, in particular when considering the entire sequence of values of the compared objects. The best known approaches to measure the similarity of time series are the Euclidean distance and Dynamic Time Warping (DTW). In some applications, the Euclidean distance produce better results whereas in other applications, DTW is superior. Thus, the distance function choice mainly depends on the application. Comparing performance of these two approaches, it is important to notice, that DTW has higher computational costs, namely $O(m^2)$, while the Euclidean distance can be computed in $O(m)$. Since we consider large databases and long time series (i.e. large values of m) in this chapter, we focus on the Euclidean distance as similarity function in the following.

Applying the Euclidean distance to the whole sequences is also only justified for short time series. Having a long time series, we face in this case two problems. The first is that the distance computation takes rather high runtimes. The second is that if the time series are indexed by a standard spatial indexing method like the R-Tree [66] or one of its variations, this index will perform rather bad because of the well-known dimensionality curse. Therefore, it is common to consider appropriate but considerably shorter approximations of the data retaining characteristic features of interest. In accordance with this schema there exist a lot of methods for dimensionality reduction leading to suitable time series representations for efficient computation of similarity distance. However, since the distances computed on the approximations do not reflect the exact similarity, they can either be used as a filter step of the Data Mining task or, if the results satisfactorily agree with the exact query response, the preliminary results can be directly taken for approximate solving the problem. In the first case, the lower bounding property should hold in the approximations to assure complete results. The second solution does not have this limitation, i.e. the approximations do not need to fulfill this lower bounding property, which makes it easier to find a proper approximation technique. More than that, since no refinements are required, the second approach will bring noticeably lower response times. The second method has, however, also a challenge: in order to achieve suitable results, the distances calculated on the approximations should accurately estimate the distances on the exact time series (i.e. approximate results of high accuracy).

After all, the question is which approximation we should use. Adequate time series approximations can be constructed by means of mathematical models. Most approaches use models based on approximations in time, i.e. models that describe how a time series depends on the time attribute (cf. Section 7.2). These techniques are characterized by the decreasing of approximation quality with increasing length of the time series (supposed a constant approximation size). We propose a method for the approximation of time series based on mathematical models that examine dependencies between various time series. Each time series is represented by an appropriate combination of a set of characteristic reference time series (normally these reference time series can be easily determined e.g. by a domain expert). This compressed representation consists of some low-dimensional feature vector that we can easily index using any Euclidean index structure. The similarity distance used for the clustering is calculated by applying the parameters that determine the combination. As a result, the costs of the clustering process depend only on the number of reference time series rather than on the length of the entire time series. As our experiment will show, even a small number of reference time series can be used to obtain rather accurate results.

Our approach is depicted in Figure 7.1. An original time series T_{orig}(shown on the upper right hand side of Figure 7.1) is approximated by an arbitrary complex combination T_{approx} of reference time series (marked as "$T1$" , "$T2$" , and "$T3$" on the left hand side of Figure 7.1) In case of Figure 7.1 this is

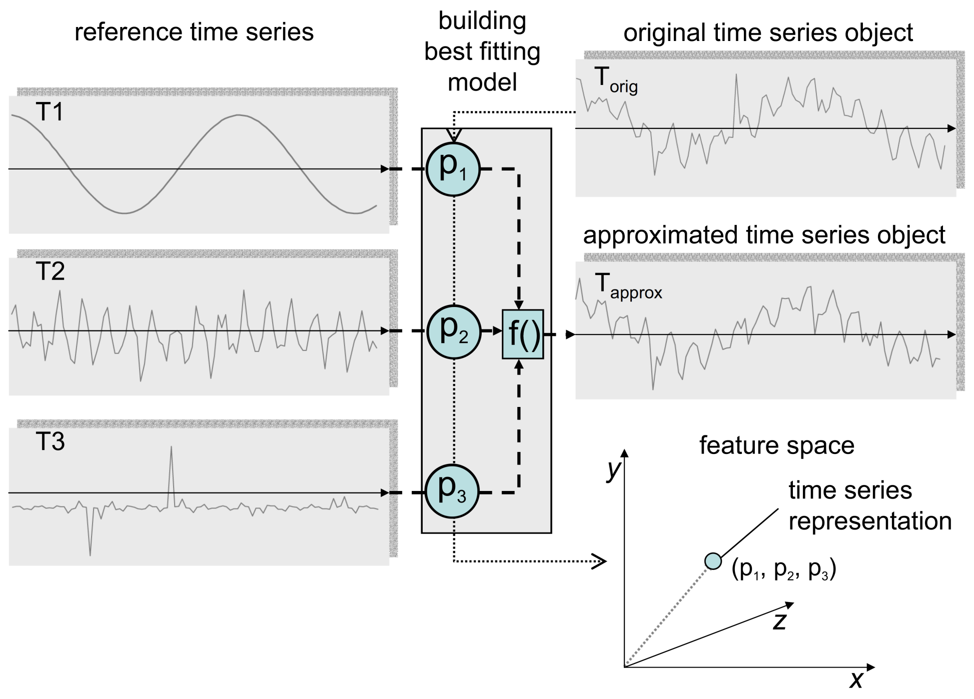

reference time series
building best fitting model
original time series object
T1
T2
T3
T_{orig}
approximated time series object
T_{approx}
p_1
p_2
p_3
f()
feature space
time series representation
(p_1, p_2, p_3)
y
z
x

Figure 7.1: Model-based time series representation

a combination of the coefficients $p_1, \ldots, p_3$ representing the three input time series using a function f. The resulting approximated time series (marked as "output" in the middle of the right hand side of Figure 7.1) is similar to the original one. For clustering, the approximation is represented by a feature vector of the coefficients of the combination (cf. lower right hand side of Figure 7.1).

Before continuing with next sections, let us give a short outline how the rest of the chapter is organized. In Section 7.2, a survey of related work is presented. In Section 7.3, after introducing the notion of mathematical models, we describe our powerful model-based method for constructing a compact representation for time series. Section 7.4 presents versatile experimental results. Finally, we conclude the chapter in Section 7.5 with a short summary and show further research directions.

7.2 Related Work

A time series of length d can generally be considered as a feature vector in a $d-$dimensional space. As mentioned above, we are to concentrate on similarity in time, in other words we suppose that the similarity of time series is expressed by the Euclidean distance of the corresponding feature vector.

As for long time series d is usually big, the efficiency and the productiveness of data analysis approaches is quite limited through the curse of dimensionality. Therefore, several more appropriate representations of time series data, for example by reducing the dimensionality, have been suggested. The majority of them are based on the GEMINI indexing method [47]: extract a few key features for each time series and map each time sequence a to a point $f(a)$ in a lower dimensional feature space, such that the distance between a and any other time series b is always lower-bounded by the Euclidean distance between the two points $f(a)$ and $f(b)$. To reach an efficient access, any well known spatial access method can be used for feature space indexing. The suggested approaches differ primarily in the representation of the time series. The latter can be classified into non data adaptive methods, including DFT [7] and extensions [201], DWT [38], PAA [205], and Chebyshev Polynomials [35], as well as data adaptive approaches, including SVD [105, 10], APCA [101] and cubic splines [16]. DFT converts a time series from a time space into a frequency space. In comparison with DFT, DWT uses not a sine function, but a so called wavelet functions. Approximation quality of DWT is a little bit better than of DFT. In PAA approach, the time series is divided into pieces of the same length whereas each piece is then approximated by one point. APCA method is similar to PAA, but uses an adaptive choice of time piece length. Chebyshev Polynomials approximation uses a linear combination of orthogonal Chebyshev polynomials. Smoothing time series with polynomials of the 3rd grade (so called splines) is the idea of cubic spline approach.

As an opposite to our approach, all these approximation techniques suppose that time series is represented by a set of attributes describing the dependency of the time series upon time. As a result, these methods are characterized by decreasing their the approximation quality with increasing length of the time series (supposing a constant number of approximation attributes).

In [162] the authors apply a clipped time series representation rather than using a dimensionality reduction technique. Each time series is represented by a bit string indicating the intervals where the value of the time series is above the mean value of the time series. By means of this representation an approximate clustering of the time series can be calculated. To speed-up the clustering task as well as to reduce the I/O cost, the bit level representations are compressed using standard compression algorithms. Unfortunately, the authors did not propose in the paper any index structure for the approximation data. Each similarity search task implies a full scan over the approximated data.

Most of the different clustering approaches for clustering time series data proposed in the past decades have been successfully applied. A common overview over clustering methods can be found in [72].

In this chapter, we claim the following contribution. We propose a novel compact approximation method for time series data that is independent of the length of the time series. The resulting representation can be indexed using any Euclidean index structure and is rather accurate for an approximate clustering of the database.

7.3 Mathematical Models for Time Series Data

Mathematical modeling is a very powerful technique for describing real-word processes by means of a compact mathematical representation (e.g., mathematical models of physical or chemical processes). In this section, after introducing a formal definition of mathematical models, we present in details our method for the description of large time series data with the help of a compact representation based on the mathematical models.

7.3.1 Mathematical Model

We begin with an informal discussion of the notion of a mathematical model. A mathematical model is an approximate description of a class of certain objects and relationships between them. This approximate description is given by mathematical formulas. Considering the notion in the context of time series data, a mathematical model describes dependencies between recorded time series data called inputs (or exploratory variables) and time series data called outputs (or dependent variables) of a process under observation. For example, we can model the relationship between the air pressure in an enclosed container w.r.t. the temperature of the surrounding environment. The monitoring of both pressure and temperature values is given in the form of time series. The values of pressure are used as values of the dependent variable (i.e. as outputs), whereas the values of temperature are used as values of the exploratory variable (i.e. as inputs). To be formal, the definition of a mathematical model can be introduced as follows.

Definition 8 (Mathematical Model) *A mathematical model $M = (V, P, f)$ for a dependent variable A (output) consists of a set of exploratory variables $a_1, \ldots, a_k (V = [a_1, \ldots, a_k])$ (called inputs) and a mathematical function $f(V, P)$ that is used to describe the dependency between the variable A and the variables $a_1, \ldots, a_k$, where P denotes the model parameters also called coefficients of the model. The general form of the model is given by $A = f(V, P) + \epsilon$, where ϵ denotes the random error.*

In this definition, the exploratory variables $a_1, \ldots, a_k$ are inputs of the model. The model parameters P represent the quantities estimated in the course of the modeling process. The random error ϵ characterizes the accuracy of "statistical" relationship between the dependent variable and the exploratory ones. In contrast to deterministic relationship, statistical means that the functional relationship holds only in average (i.e., not for each data point).

Generally, to construct a mathematical model for a time series A of measured values as a dependent variable, we need a mathematical function f and a set $V = a_1, \ldots, a_k$ of input time series also called *reference time series*. f and V can usually be given by a domain expert or can alternatively be chosen by exploring a small sample of the time series in the database. Our purpose is to find the "best fitting" model, i.e. the model for which the random error ϵ is minimal. This minimization can be obtained by calculating appropriate model parameters P. In the recent decades, a few methods were proposed allowing us to fit the model to the real time series data (i.e. to calculate the model parameters P so that the random error ϵ is minimized). The most popular method is Least- Squares Estimation which we will use in the following.

At first, consider some examples of mathematical functions that are commonly used in mathematical modeling. For a time series A that fits a straight line with an unknown intercept and slope, there are two parameters $P = (p_1, p_2)$, and one exploratory variable V such that $f(V, P) = p_2 \cdot V + p_1$.

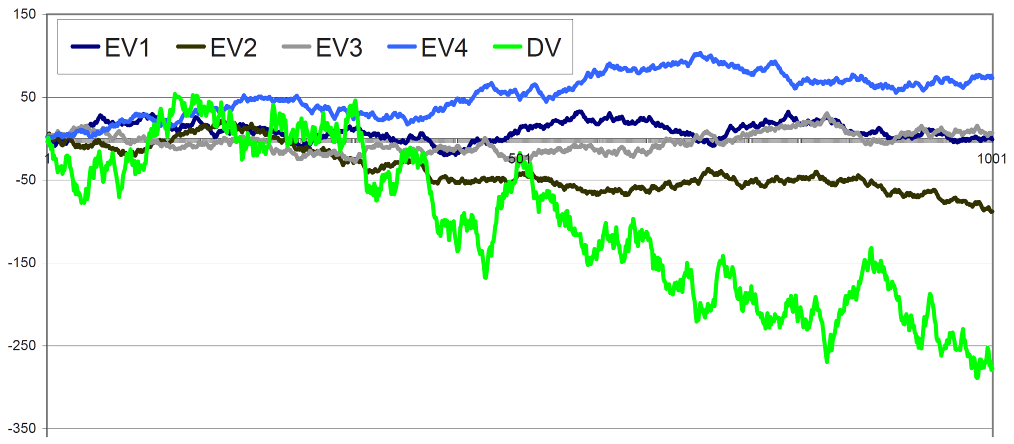


Figure 7.2: An example for relationship between a dependent variable (DV) and four exploratory variables (EV1-4)

In Figure 7.2 we have an illustration for the approximation of a more complex time series $A = DV$ by means of a mathematical model with a combination of four reference time series $a_1 = EV1$, $a_2 = EV2$, $a_3 = EV3$, and $a_4 = EV4$ expressed by $DV = EV1 + 2 \cdot EV2 - 4 \cdot EV3 - EV4$.

So, the mathematical model describing $A = DV$ is composed of the set of reference time series $V = \{a_1, a_2, a_3, a_4\}$ and a function $f(V, P) = a_1 + 2 \cdot a_2 - 4 \cdot a_3 - a_4$ and $p_1 = 1, p_2 = 2, p_3 = -4, p_4 = -1$.

Summarizing all this, a mathematical model gives an elegant technique of describing the relationship between a dependent variable (output time series) and a set of exploratory variables (reference time series). Any complex mathematical function, like a combination of quadratic and logarithmic functions, can be used to approximate this relationship. Of course, if we want to express the relationship formally, we need to fit the parameters of a given mathematical function.

7.3.2 Representation of Time Series Based on Mathematical Models

This section is dedicated to presenting the intuition behind our compact time series representation and to the introduction a novel technique which can translate even a very long time series into a compact representation.

Let us consider a given set of reference time series V and a given mathematical model $M = (V, P, f)$. We can now consider each time series $T_i \in \mathcal{D}$ in the database as a dependent variable a_i. The values of outputs a_i can be approximated by values of the mathematical model $M_i = (V, P^i, f)$ containing the model parameters P^i that are fitted in order to approximate the values of the dependent variable a_i as close as possible. Thus, by means of the model parameters P_i, the given mathematical model M_i expresses relationships between the reference time series V and the approximated time series T_i (that is it describes how strong is dependency of a_i upon each of the reference time series). Evidently, dependent variables a_i and a_j with similar dependencies should have also similar mathematical models M_i and M_j, i.e. rather similar will be the parameters P^i and P^j. In other words, if the underlying physical processes represented by measured values in the database have similar character, their mathematical models look very similar. The justification of this relation between original processes and mathematical models is caused by the fact, that dependencies we examine are based on the same form of the mathematical function and the same reference time series. That is, all our models M_i use the same function f and the same set of reference time series V, distinguishing only in the parameters P^i.

This intuition can be expressed more formally as follows:

Definition 9 (Model-based Representation) *Let $V = a_1, \ldots, a_k \subseteq \mathcal{D}$ be a given set of reference time series with $a_j = \langle a_{j,1}, \ldots, a_{j,N} \rangle$ and let $f(V, P)$ be a given mathematical function. A model-based representation of a database time series $T_i = \langle t_{i,1}, \ldots, t_{i,N} \rangle \in \mathcal{D}$ is given by a vector of model parameters P^i if P^i minimizes the random error ϵ of the mathematical model $M = (V, P, f)$ having the general form $T_i = f(V, P^i) + \epsilon$*

In the example shown in Figure 7.2, the model-based representation of the time series DV w.r.t. the reference time series $\rho = EV1, EV2, EV3, EV4$ is expressed by a vector $\alpha = (1, 2, -4, -1)$. It should be noted, that in this case the description of a time series of length 1,000 is reduced to a compact model-based representation with four coefficients.

To summarize, we describe each time series by a small set of model parameters of a mathematical model the form of which is the same for all time series in the database. The size of our model-based representation does not depend on the length of the time series in the underlying database but depends only on the number of reference time series. Particularly, how exact the approximation of our model-based representation is, solely depends on the applied model function and the reference time series. This provides us a possibility to achieve the desired level of exactness of the approximation. This can be realized by choosing a model function and a set of reference time series which are most suitable for the given application area.

7.3.3 Model-Based Similarity of Time Series

The determining factor in time series clustering is the distance (or similarity) measure used to make a decision about the similarity of time series. As already mentioned above, we concentrate on similarity in time. Thus, for our method we use the best known time-based distance measure for time series, the Euclidean distance. This distance is commonly used for the dimensionality reduction techniques noticed in Section 7.2.

For long time series, to calculate the Euclidean distance is very expensive. More than that, the efficiency of indexing methods, used to speed-up similarity queries, is restricted by the the well-known curse of dimensionality. To avoid this, we propose to calculate the similarity using the representations based on mathematical models consisting of only a few coefficients (model parameters). Similarity distance based on the model parameters, as we can show, accurately approximates the Euclidean distance between the original time series. The approximation accuracy chiefly depends on how good the model fits to the original time series, i.e. how exact the model approximates the original time series. While determining the similarity of time series based on the model parameters, we need to take into account that the pairwise similarities between our reference time series are not necessarily be identical. An illustration for this is shown in Figure 7.3. A model M based on the three reference time series presented at the top of Figure 7.3 represents the depicted below three time series T_1, T_2, and T_3. T_1 equals the first reference time series, that is why the coefficients of its model-based representation are given by $P^1 = (1.0, 0.0, 0.0)$. Similarly, since T_2 is equal to the second reference time series, the coefficients of the model-based representation of T_2 are expressed by $P^2 = (0.0, 1.0, 0.0)$. Almost the same picture is finally with T_3

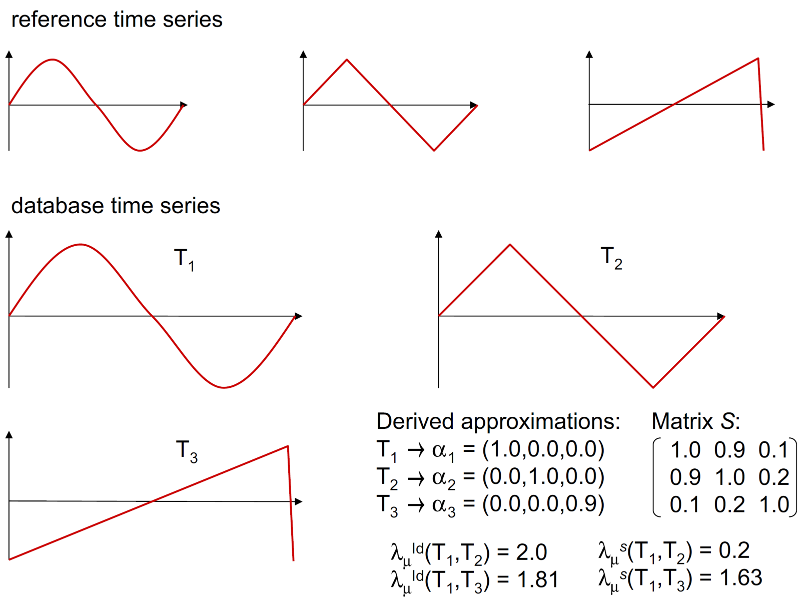


Figure 7.3: Motivation for the use of the Mahalanobis-distance

and its model-based representation. T_3 is nearly equal to the third reference time series, so the coefficients of its model-based representation are given by $P^3 = (0.0, 0.0, 0.9)$. Comparing the Euclidean distance between T_1 and T_2 (in the Figure it is denoted by $\lambda_M^{Id}(T_1, T_2)$) with the Euclidean distance between T_1 and T_3 (denoted by $\lambda_M^{Id}(T_1, T_3)$) we notice that $\lambda_M^{Id}(T_1, T_2) > \lambda_M^{Id}(T_1, T_3)$.

This is quite unintuitive since there are much more similarities between the original time series T_1 and T_2 than between T_1 and T_3. The reason for this is the fact that, when we working with reference time series, we do not take into account that the first reference time series is more similar to the second than to the third one. Thus, when calculating the similarity between the model parameters P^i and P^j of two time series $T_i, T_j \, in \, \mathcal{D}$, it is needed to consider these different pair-wise similarities of the reference time series. We can do this by means of the well-known *Mahalanobis-distance* [123] between the vectors P^i and P^j, formally

Definition 10 (Model-based Similarity Distance) *Let $T_i, T_j \in \mathcal{D}$ be two time series and let $M = (V, P, f)$ be a mathematical model where P^i and P^j*

are the representations of T_i and T_j based on M, respectively. The model-based similarity distance $\lambda_M^{Id}(T_i, T_j)$ between T_i and T_j is defined by

$$\lambda_M^{Id}(T_i, T_j) = \sqrt{(P^i - P^j) \cdot S \cdot (P^i - P^j)^T}$$

The key role in the Mahalanobis-distance plays the matrix S which is used to rank the pair-wise combinations of the reference time series. So, to determine an appropriate matrix S for distance computation is a significant task. The following lemma assists in this choice.

Lemma 11 *Let $M = (V, P, f)$ be a mathematical model and P^i and P^j be the representations of time series T_i and T_j based on M, respectively. Then, the values of the model-based similarity distances are approximately equal (except for a small random error Δ) to the values of Euclidean distance on the original time series T_i and T_j , i.e. $Dist_{Euclidean}(T_i, T_j) \approx \lambda_M^{V \cdot V^T}(T_i, T_j)$ or $Dist_{Euclidean}(T_i, T_j) = \lambda_M^{V \cdot V^T}(T_i, T_j) + \Delta$.*

Proof. *Without loss of generality, we suppose that the function f is linear or can be transformed to a linear form, i.e. $T_i = P^i \cdot V + \epsilon_i$ for any $T_i \in \mathcal{D}$.*

$$Dist_{Euclidean}(T_i, T_j) = \sqrt{(T_i - T_j) \cdot (T_i - T_j)^T} =$$
$$\sqrt{((P^i \cdot V + \epsilon_i) - (P^j \cdot V + \epsilon_j)) \cdot ((P^i \cdot V + \epsilon_i) - (P^j \cdot V + \epsilon_j))^T} =$$
$$\sqrt{((P^i - P^j) \cdot V) \cdot ((P^i - P^j) \cdot V)^T + \Delta'} =$$
$$\sqrt{((P^i - P^j) \cdot V) \cdot (V^T \cdot (P^i - P^j)^T) + \Delta'} =$$
$$\sqrt{((P^i - P^j) \cdot (V \cdot V^T) \cdot (P^i - P^j)^T) + \Delta'} =$$
$$\lambda_M^{V \cdot V^T}(T_i, T_j) + \Delta$$

The lemma claims that the model-based similarity distance approximates the Euclidean distance on the original time series[1] by an error of Δ, if $S = V \cdot V^T$ where V is a matrix composed of the reference time series. Obviously, Δ depends on the random errors ϵ_i and ϵ_j of the model-based approximation of T_i and T_j . Thus, if the errors of the approximation are small (which

[1]Please recall that we consider similarity in time instead of similarity in shape, and, thus the Euclidean distance is used as a baseline.

is a design goal of the approximation and is realized by the Least-Squared Error method), then Δ will also be small. Hence, setting $S = V \cdot V^T$, the model-based similarity distance gives us a quite accurate approximation.

It is significant, that if we have the unity matrix as S, i.e. $S = V \cdot V^T = Id$, the Mahalanobis-distance is identical to the Euclidean distance. In Figure 7.3, we compare the model-based similarity distance (between T_1 and T_2 as well as between T_1 and T_3 using $S = V \cdot V^T$ denoted by $\lambda_M^S(T_1, T_2)$ and $\lambda_M^S(T_1, T_3)$, respectively) with the corresponding Euclidean distance on the model-based representations. As it can be observed, the model-based similarity distance in the form of the Mahalanobis-distance more accurately reflects the intuitive similarity of the original time series than the Euclidean distance on the model-based representations.

More than that, it is important to note that our approach may have a slight rise of the CPU cost due to use of the Mahalanobis-distance instead of the Euclidean distance used by the existing methods. However, since we are dealing not with the original time series but with its compact representation, the size of which is independent of the length of the original data items, this minor loss of CPU performance does not play any significant role because we get a great benefit in terms of I/O-cost especially when dealing with long time series.

7.3.4 The Choice of the Reference Time Series

The choice of the reference time series is, of course, an important aspect of our model-based representation of time series. As already outlined, this selection can usually be done by a domain expert, assumed that we have one at hand. For the case, when we have no such domain expert at hand, we need a procedure for the choice of the reference time series.

Generally, there should be a high correlation between the reference time series and a subset of the remaining time series in the database. Inspired by this insight, our suggestion is to use the following procedure to derive a set of reference time series. Supposed we want to choose k reference time series, we simply cluster the time series using a $k-$medoid clustering algorithm, e.g. PAM [72]. As a result we have a set of k cluster medoids (time series), each of which represents its corresponding cluster. All time series of a cluster strongly correlate with the corresponding cluster medoid. Thus, it must be a good choice to take these medoids for the derivation of the reference time series. Additionally, in order to reduce computational costs, we suggest to apply the PAM clustering only on a small sample of the database. A sample rate of about 1%, as shows practice, provides a sufficient high clustering accuracy.

7.3.5 Efficient Approximative Clustering

Now, using the similarity distance measure defined above, it is possible to apply any analysis task to time series data. Our chief goal is to get an efficient clustering of the database $\mathcal{D}$ of time series based on the approximative representations, at the same time keeping the quality of generated clusters at sufficient level. Thus, approximative clustering enables obtaining very fast response times while accepting a considerable accuracy reduce. This appears to be reasonable in many applications domains. Although we used in our experiments the most known clustering method k-means, our approximate representation can be easily integrated into any other clustering algorithm the user is most comfortable to work with. The main issue for approximate clustering is without doubt to generate finally accurate results, i.e. the used approximations should describe the original time series appreciably well. As we will show experimentally, even a very small set of parameters for the time series approximations suffices to achieve high quality clustering results. This is still true if the original time series that should be clustered are very long. An illustration indicating the potentials of our approximation is presented by an example in Figure 4. Here we compare two quite complex time series T_1 and T_2 of length 1,024 with a few sample approximations using just 44 coefficients. The approximation of the real time series given by the compressions, as we can see, is quite accurate.

7.4 Evaluation

To provide insight into the performance of our method compared to existing ones, we implemented our approach and comparison partners in Java 5. We carried out our experiments on a workstation featuring 2x3GHz Xeon CPUs and 32GB RAM. We used four datasets, three of which are artificial (DS1, DS2 and DS4) and one real world dataset (DS3). These are shown in Table 7.1.

For the first two artificial datasets - DS1 and DS2, we used random walk to generate the reference time series. The corresponding datasets represent a

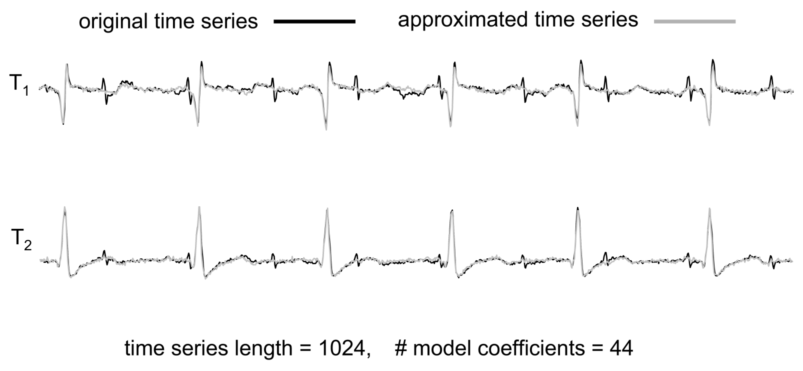


Figure 7.4: Approximations for sample time series

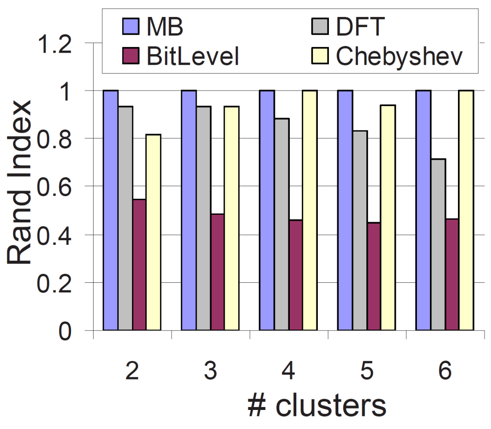
Name	Type	Length
DS1	artificial	2,560
DS2	artificial	6,000
DS3	real world	1,024
DS4	artificial	2,000 - 14,000

Table 7.1: Real and Artificial Data Sets with.

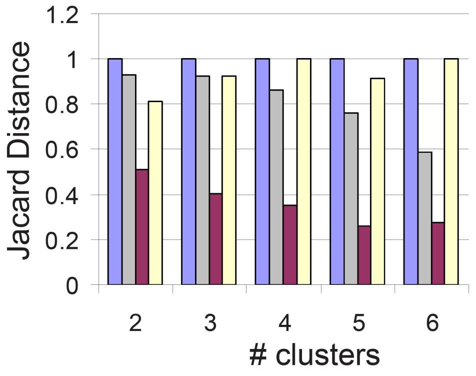
linear combination of the reference time series compounded by the identity, square, cube and first and second derivatives.

In order to demonstrate that our approach can handle versatile data, the third artificial dataset DS3 we composed as follows. It consists of real-world time series from the following application areas: (1) wing flutter[2], (2) cutaneous potential recordings of a pregnant woman[3], (3) data from a test setup of an industrial winding process[4], (4) continuous stirred tank reactor[5]. In this dataset (DS3), the reference time series we get from domain experts. To generate the forth dataset DS4, we use Cylinder-Bell-Funnel method[6]. Here we are dealing with an artificial dataset that covers the entire

[2]`http://homes.esat.kuleuven.be/~smc/daisy/daisydata.html`
[3]`http://www.tsi.enst.fr/icacentral/base_single.html`
[4]`http://homes.esat.kuleuven.be/~smc/daisy/daisydata.html`
[5]`http://www.fceia.unr.edu.ar/isis/cstr.txt`
[6]`http://waleed.web.cse.unsw.edu.au/phd/html/node119.html`



(a)



(b)

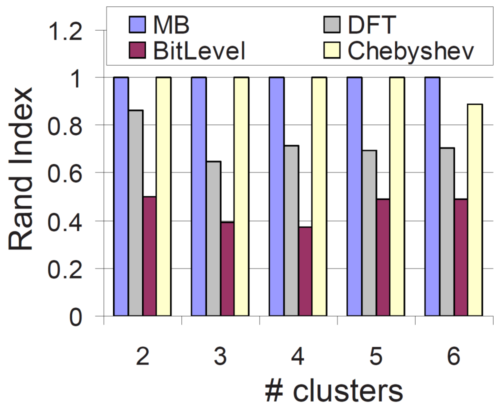
Figure 7.5: Cluster quality for varying number of clusters (DS1)

spectrum of stationary/nonstationary, symmetric/asymmetric, cyclical/non-cyclical, noisy/smooth, etc. data characteristics. To get the reference time series, we used a PAM clustering ($k = 4$) of a random sample as described before. To determine the k parameter we use standard methods.

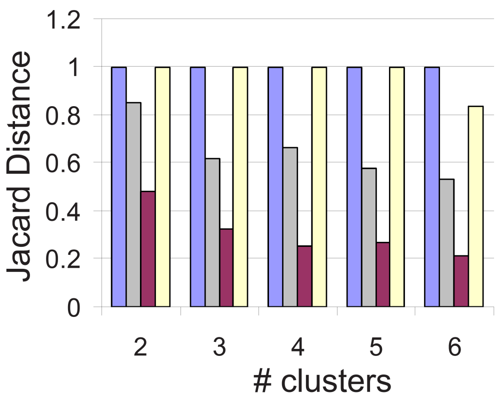
Now we make comparison between our mathematical model based time series approximation (MB) and existing approximation techniques. As competitors we consider the following approaches: Bit Level using clipped time series representations as proposed in [162], Discrete Fourier Transformation (DFT) [7] and representations by means of Chebyshev polynomials (Chebyshev) [35]. The evaluation of the competing techniques is made using the approximation quality of k-means clusterings.

Model description of the test datasets. For the mathematical function of the model we used a linear combination of the original set of reference time series, the quadrature and cubature of the reference time series, as well as the first and second derivation of the reference time series in time. In fact, using n model parameters we finally required only $n/5$ reference time series.

Measuring clustering quality. To carry out the experimental evaluation of our method, we constructed reference clusterings based on the Euclidean distance between the original time series and measured the clustering quality



(a)



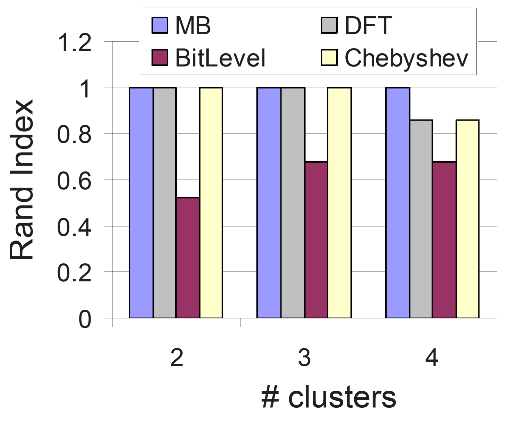
(b)

Figure 7.6: Cluster quality for varying number of clusters (DS2)

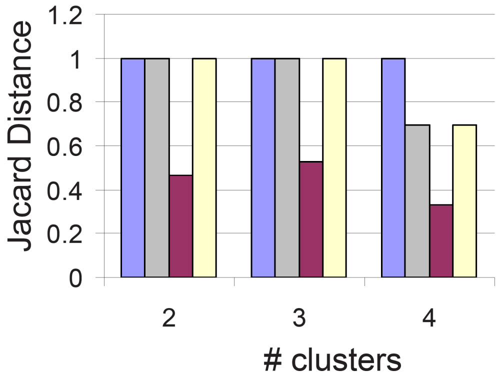
w.r.t. this reference clustering. To measure clustering quality, we used the two most known (for the year 2008) clustering evaluation methods, namely the Rand Index and the Jacard Distance [68]. These methods are intuitive, at that Rand Index is especially good for a small number of clusters whereas Jacard is good for a big number of clusters.

Experiments on clustering quality. In our first experiment, we look closer at the quality of our approximation method for a varying number of clusters based on the three datasets DS1 (cf. Figure 7.5), DS2 (cf. Figure 7.6) and DS3 (cf. Figure 7.7). Over all competitors, the Bit Level approach shows the lowest clustering quality for all experiments and experimental settings. The quality we obtain with our method is at least two times higher than that of the Bit level approach. In our experiments, our approach outperforms the method based on DFT coefficients and is even gives better results than the approach using the Chebyshev polynomials when increasing the number of searched clusters.

In comparison with the competitors, our approach achieves optimal clustering quality, also on the real world dataset. The explanation for this is the fact that our model based similarity distance reflects the Euclidean distance on the original time series very exactly.

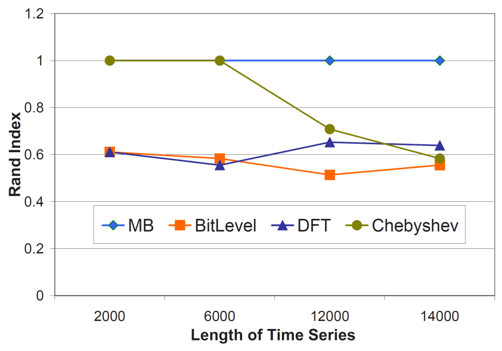


(a)

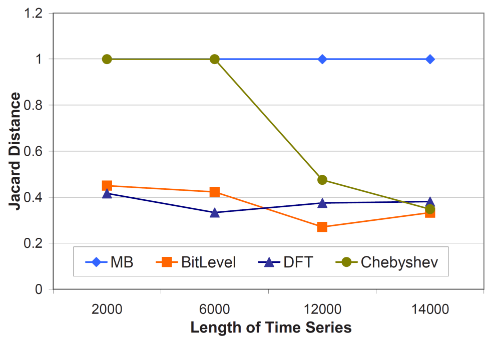


(b)

Figure 7.7: Cluster quality for varying number of clusters (DS3)

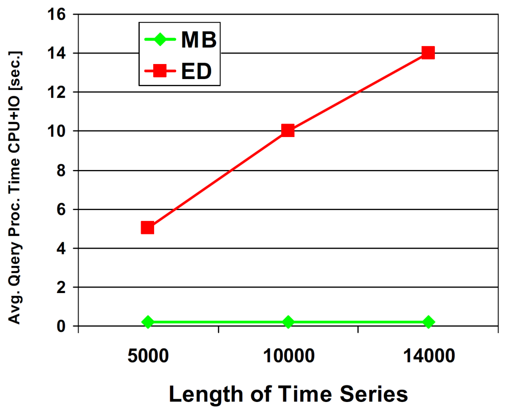


(a)

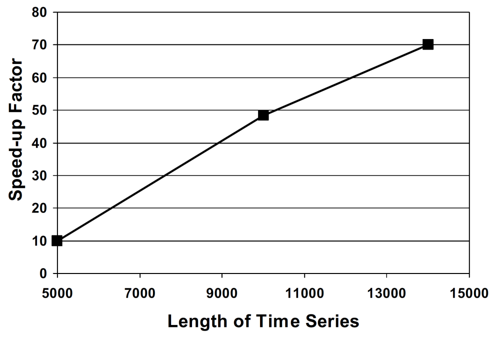


(b)

Figure 7.8: Cluster quality for varying time series length (DS4)



(a)



(b)

Figure 7.9: Performance of our model-based approach vs. Euclidean distance

Dependency on time series length. In the next experiment, we answer the question how the cluster quality depends on the size of the time series. Figure 7.8(a) and Figure 7.8(b) illustrate the results. As it can be seen, the characteristic of both approaches DFT and Chebyshev, both based on dimensionality reduction, is that the clustering quality decreases drastically with increasing time series length. In contrast, using our model-based approximations we achieve high quality over all investigated time series lengths. Similar to our method, the Bit Level approach keeps nearly constant quality even for long time series, having however low performance.

Runtime comparison. In our last experiment, we compare the speed-up of our approach with respect to the original Euclidean distance in terms of CPU time. For that purpose, we varied the length of the time series of DS4. The results are depicted in Figure 7.9(a). As it can be expected, our model-based method (marked with "MB"in the figure) scales constant, while the Euclidean distance (marked with "ED" in the figure) scales linear w.r.t. the length of the time series. It can also be seen, for long time series our model-based approach clearly outperforms the Euclidean distance. Figure 7.9(b) illustrates the speed-up factor, which gains our model-based approach over the Euclidean distance based one. We can observe, this speed-up grows linearly when increasing the length of the time series. As result, our approach yields feature vectors of a constant and considerably lower dimensionality and (beside more efficient indexing) yields better CPU performance than using the original time series.

7.5 Conclusions

The performance of important Data Mining task like clustering time series is primarily limited by the length of the given time series. Recent research has proposed several dimensionality reduction approaches to derive a compact approximation of time series. Approximative clustering using existing compressed representations based on dimensionality reduction, usually tends to low accuracy. Especially, this concerns large time series.

In this chapter, we propose an approximation technique for time series based on the idea of mathematical models. According to this approach, each time series is described by the coefficients of a mathematical model involving a given set of reference time series. The great advantage thereby is that the size of our approximation depends only upon the number of coefficients of the model (in other words, upon the number of reference time series). This means that our method is independent of the length of the original time series and in particular can be applied also for very long time series. The resulting compressed representation using a feature vector of coefficients of the model allows efficient indexing of the time series approximations for fast similarity search and clustering. We further show how our proposed approximations can be used for approximate clustering. As we evaluated experimentally, our novel approach outperforms existing state-of-the-art approximation methods in terms of clustering accuracy, i.e. our approximations are considerably better than existing schemata.

In the future work, we are going to extend our ideas of approximating time series by means of mathematical models to stream data.

Chapter 8

Model-based Classification of Data with Time Series-valued Attributes

Similarity search and Data Mining on time series databases has recently attracted much attention. In this chapter, we represent a data object by several time series-valued attributes. Although this kind of object representation is very natural and straightforward in many applications, there has not been much research on Data Mining methods for objects of this special type. In this chapter, we propose a novel model-based classifier exploiting the benefits of representing objects by several time series. Classification decisions are based on class-specific interaction patterns among the time series of an object. Experimental results on benchmark data and real-world medical data demonstrate the performance of our classifier.
Parts of the material presented in this chapter have been published in Paper [24]. In this paper, Andrew Zherdin has developed the application of the mathematical models for classification. Andrew Zherdin's part was also the model optimization and carrying out the experiments. Claudia Plant has proposed the interpretability of the models and optimized the experiments. She has also proposed to take BIC for the dimension choice. Afra Wohlschläger and Leonhard Läer have done the medical part of the work.

8.1 Introduction

In the last chapter, we used mathematical models for approximation task when dealing with univariate time series. The choice of reference time series is a non-trivial and data dependent step. Many real-word objects are represented however as multivariate time series. In this chapter, we will have to do with multivariate time series and apply mathematical models to classify them. The models building algorithm proposed here does not need searching for reference time series.

There is a huge volume of papers on classification of time series, e.g. [93, 99, 127]. However, most existing approaches consider a single time series as the data object to be classified. In this chapter, we focus on the representation of a data object by several time series-valued attributes and propose a novel algorithm for classification of such objects. Representing an object by a set of time series appears very natural and straightforward in many applications.

Consider for example a meteorological database storing measured data from different meteorological stations. Each station can be regarded as an object which is represented by several time series-valued attributes, including temperature, rainfall and air pressure. Based on the measured data over a certain period, stations can be automatically classified e.g. into stations with normal weather conditions and stations with exceptional conditions which are presented to a human expert.

In many medical applications, it is natural to represent a data object by a system of time series. Consider for example EEG data: Electroencephalography measures the electric activity in the brain by an array of usually 64 electrodes which are distributed over the scalp [145]. It is very natural to consider the EEG of one person as a data object, where the 64 time series-valued attributes represent the 64 electrodes. Another example is functional magnetic resonance imaging (fMRI) [65]. This imaging modality yields time series of 3-dimensional images of the brain. Also in this case, it is reasonable to regard each subject as a data object which is represented by time series-valued attributes. Each 3-d pixel called voxel of the fMRI image is an attribute of time series type.

Increasing amounts of captured motion stream data are collected in multimedia applications [34]. Gesture sensing devices, such as a CyberGlove usually contain multiple sensors to capture human movements. Motion classification is the task to automatically assign movements, usually performed by different people, into predefined motion classes and has important applications in gait analysis and virtual reality. Again, it is reasonable to regard each movement as a data object which is represented by the time series obtained from the different sensors of the gesture sensing device.

Although representing an object by a system of time series is very natural in many applications, there has not been much research on Data Mining methods for objects of this special type. We propose a classifier which exploits the benefits of this kind of object representation by focusing on interaction patterns between the time series within a data object. We derive a description of each class in terms of a system of linear models characterizing class-specific interaction patterns. Our experiments demonstrate that the information on interactions substantially improves the classification accuracy. In addition, the models provide interesting insights for understanding class-specific differences.

The remainder of this chapter is organized as follows. After a brief survey of related work in the next section, we introduce our classifier in Section 8.3. Section 8.4 presents an experimental evaluation and Section 8.5 concludes the chapter.

8.2 Related Work

To describe the interaction patterns within the time series of an object, we follow a model-based approach first introduced in the Chapter 7 and the paper [106] for the purpose of efficient compression of time series. To derive an approximation to speed up similarity search, in this approach each time series is represented by a combination of a set of specific reference time series. We use equivalent linear models to represent interactions of time series and a similar algorithm for model finding, but with a very different objective.

In comparison to general time series classification, only relatively few papers focus on classification of multivariate time series. In [204] a decision tree for data with time-series attributes is proposed. At the internal nodes of this tree, time series are stored and splits are performed based on dissimilarity between pairs of time series. The method selects upon split a time series which exists in data by exhaustive search based on class and shape information.

In [94] the authors present tClass, an algorithm for classifying multivariate time series data based on so-called meta-features. These meta-features model some kind of recurring substructure in the instances, such as strokes in handwriting recognition, or local maxima in time series data. The types of substructures are defined by the user, but the substructures are extracted automatically and are used to construct attributes. The parameter space defined by the meta-features is segmented into regions beneficial for classification using a heuristic technique. After feature construction, classification is performed with conventional propositional learning algorithms, such as decision trees.

For EEG data, some specialized classifiers have been proposed, e.g. in [145] to use EEG as a biometric for person identification. To identify persons with high accuracy based on EEG data with visual stimuli, after selection of the best discriminative channels classification is performed by a neural network.

Also in [146] a classification method especially designed for EEG with visual stimuli is proposed. This method extracts the visual evoked potential (VEP) which is embedded in the ongoing EEG and blurred by noise. The VEP contained in the gamma band is extracted by digital filters. Similar to [145], classification is performed by a neural network.

In [206] a distance-coupled Hidden Markov model (HMM) is introduced for classification of EEG data. The distance-coupled HMM applies several HMMs to model multivariate time series. The link between the different HMMs is established by the fact that the state of one model at some particular time stamp t depends on the states of the other models at time stamp $t-1$. In their experiments the authors demonstrate the usability of coupled HMMs for EEG classification. In contrast to [206], our models capture dependencies not between time points but between the dimensions. Our models are applicable to classes and not to single objects. So our models have a very different structure compared to HMMs.

8.3 Model-Based Classifier

Our method for model-based classification involves 3 steps:

1. model finding,
2. model refinement,
3. classification.

In the training phase of the classifier, in the first step the interaction patterns of the time series within each class are characterized by a system of linear models. In the second step, these models need to be refined to be suitable for classification. The final step is the classification of test objects.

Notations and Definitions. We represent each class as a system of linear models. Following [106], a single model can be defined as follows.

Definition 12 (Model) *A mathematical model $M = (V, P, f)$ for a dependent variable A (output) consists of the following parts:*

- *a set of* exploratory variables $a_1, \dots, a_m$ *(inputs),*
- *a* mathematical function

  $$f(V, P)$$

  that is used to describe the dependency between A and $a_1, \dots, a_m$. P denotes the model parameters also called coefficients of the model.

The general form of the model is given by $A = f(V, P) + \varepsilon$, where ε denotes the random error.

Note, that Definition 12 is the same as Definition 8 in Chapter 7. For better readability we repeat it here again.

In this chapter, we focus on linear models, i.e. f is a linear combination of the inputs.

Further, we consider a data set DS with n objects, i.e. $DS = \{O_1, ..., O_n\}$. Each object O_i is represented by d time-series valued attributes a_j, each consisting of a real-valued time series with m time points, i.e. $O_i = \{a_1, ...a_d\}$ and $a_j = \{t_1, ..., t_m\}$. For the training data, a categorical class label is known. To facilitate our argument, we focus on a 2-class problem with class labels *class1* and *class2*. However, our method can be straightforwardly applied in a multi-class setting.

Model Finding. To capture the interaction patterns of time series within a class, e.g. *class1* we first have to generate a summarization of the time series attributes of the objects of the class. For each of the d dimensions, we concatenate the time series of all objects of the class. As a result we have a novel object O_s with d time series-valued attributes with $|class1| \cdot m$ time points. Now, we can derive a system of linear models as specified in Definition 12 for O_s. All d attributes of O_s are applied as output A once. The idea is to try to explain each individual attribute by a linear combination of (potentially all) other attributes. To avoid over fitting, models with few inputs are preferable. In addition, an exhaustive search for all combinations of inputs is not feasible for large d. To cope with these problems, stepwise algorithms are often a good choice [110, 57].

We propose an iterative stepwise algorithm for model finding which is displayed in pseudocode in Figure 8.1. The algorithm has two parameters: the time series being modelled (output) and a set of time series (inputs) modelling the output time series. Our algorithm computes a model M for the given output A and inputs V time series and returns this model as result. Our purpose is to find a subset of V containing only those inputs which are most relevant to explain A. To achieve a balance between model complexity and goodness of fit, we apply the Bayesian Information Criterion (BIC) [57]. Theoretically well founded in Bayesian statistics and information theory, this criterion penalizes overly complex models and allows us to select the most relevant inputs in a parameter-free way. Our algorithm starts with an empty set $relevantInputs$. It then iteratively either adds a new input to $relevantInputs$ or removes an already existing element depending on which of these two actions leads to a greater improvement of BIC (1). Based on these $relevantInputs$ the model is constructed by least-squares fitting. The algorithm terminates if no further improvement of BIC can be achieved (2). The algorithm finally returns the model based on $relevantInputs$ (3).

Model Refinement. After application of the model finding algorithm using each attribute of O_s as an output, we obtain a set of models SM consisting of d models per class. However, these models can not be directly applied for classification. SM may contain models with large error and models representing general trends in the data which are not class-specific. Therefore the set of models needs to be refined to build a classifier with good discriminatory power. As depicted in pseudocode in Figure 8.2, the algorithm for model refinement involves two steps: First, we exclude models with large error since they do not capture any distinct interaction information. This is implemented by removing all models having an error larger than three times the median of the error on training data (1). From the remaining models we select those which discriminate between classes. For all classes and models, we compute the cumulative error on the training data. Models having a smaller cumulative error for the correct class than for the wrong class are

```
algorithm findModel(Output $A$, Inputs $V$): Model $M$
  $relevantInputs := \emptyset$;
  $oldMIN := +\infty$;
  do {
//(1) find a relevant input
      $[minPlus, x^+] := \min_{x \in (V \setminus relevantInputs)} (BIC(relevantInputs \cup \{x\}, A))$;
      $[minMinus, x^-] := \min_{x \in relevantInputs} (BIC(relevantInputs \setminus \{x\}, A))$;
      if($\min(minPlus, minMinus) > oldMIN$) then
// (2) no more relevant inputs
        break;
      end if
      if($minPlus < minMinus$) then
        $relevantInputs.add(x^+)$;
        $oldMIN := minPlus$;
      else
        $relevantInputs.remove(x^-)$;
        $oldMIN := minMinus$;
      end if
  } while()
// (3) build the Model with inputs
  $M := new\ Model(relevantInputs, A)$;
return $M$;
```

Figure 8.1: Algorithm for Model Finding. A and V are parameters of $findModel$. The model M is the result of the method.

included in the classifier (2).

A and V are parameters of $findModel$. The model M is the result of the method.

Classification. The classification of the test objects is now simple. To classify an object, we sum up the mean square error for all relevant models for all classes. We assign the object to the class with the smallest mean square error.

Runtime Complexity. The worst-case runtime complexity of the step-wise model finding algorithm is quadratic in d. Usually, the number of time points m is much larger than the number of dimensions d. Due to matrix inversion required for least square model fitting, we then have cubic complexity in m. The model refinement step, as well as the classification of an object are quadratic in d and linear in m.

8.4 Evaluation

Data Sets. Table 8.1 provides a summary on the data sets used for evaluation. Data sets 1 to 8 are different EEG data sets available at the UCI machine learning repository[1]. These data sets are derived from a study on alternations of brain activity in alcoholic subjects in comparison to a control group. Each subject was exposed to either a single visual stimulus (S1) or to two visual stimuli (S1 and S2). When two stimuli were shown, they were presented in either a matched condition where S1 was identical to S2 or in a non-matched condition where S1 differed from S2. Data sets 1 to 4 correspond to the so-called Large Data Set at UCI where training and test data are available. The task is to classify the subjects into the classes alcoholic and control based on 10 time series-valued attributes. To generate a representation of the objects of a class, we concatenated the time series of all objects and all runs resulting in a time series of 2,560 time points. Data sets 5 to 8 are derived from the so-called Small Data Set which contains data of 2 subjects, one alcoholic and one of class control. For each of the 3 experimental paradigms, 10 runs of EEG have been recorded. The task here is to classify the EEG runs to the classes alcoholic and control.

[1]http://archive.ics.uci.edu/ml/databases/eeg/eeg.data.html

```
algorithm refineModels(set of models $SM$, training objects $O$): refined set of
models $RSM$
// (1) remove corrupt models
  $median$ := median error of $SM$;
  for each $i \in \{1 \ldots SM.size()\}$ do
    if($SM[i].error > 3 \cdot median$) then
      $SM.remove(i)$;
    end if
  end for

// (2) find class separating models

  for each $m \in \{1 \ldots SM\_class1.size\}$ do
    for each $o \in \{1 \ldots obj\_class1.size\}$ do
      $difference[i] += mean\_square\_error(m, o)$;
    end for
    for each $o \in \{1 \ldots obj\_class2.size\}$ do
      $difference[i] -= mean\_square\_error(m, o)$;
    end for
  end for
  for each $m \in \{1 \ldots SM\_class2.size\}$ do
    for each $o \in \{1 \ldots obj\_class1.size\}$ do
      $difference[i] -= mean\_square\_error(m, o)$;
    end for
    for each $o \in \{1 \ldots obj\_class2.size\}$ do
      $difference[i] += mean\_square\_error(m, o)$;
    end for
  end for

// result

  for each $i \in \{1 \ldots difference.size\}$ do
    if($difference[i] < 0$) then
      $RSM.add(i)$;
    end if
  end for
return $RSM$;
```

Figure 8.2: Algorithm for Model Refinement.

Name	Description	Data Objects	#Objs	#Cls	#Dims	Length
DS1	EEG single stimulus	subjects	40	2	64	≈2560
DS2	EEG matched stimulus	subjects	40	2	64	≈2560
DS3	EEG non-matched stimulus	subjects	40	2	64	≈2560
DS4	EEG combined	subjects	40	2	64	≈7680
DS5	EEG single stimulus	runs	20	2	64	256
DS6	EEG matched stimulus	runs	20	2	64	256
DS7	EEG non-matched stimulus	runs	20	2	64	256
DS8	EEG combined	runs	20	2	64	768
DS9	fMRI	subjects	26	2	90	325
DS10	motion stream	signs	2565	95	22	≈57

Table 8.1: Multivariate Time Series Data Sets.

DS9 [65] consists of 26 functional MRI images. This data set has been obtained from a study on somatoform pain disorder and consists of images of 13 diseased subjects and 13 healthy controls. For classification we selected 90 regions of interest as proposed in [190].

DS10, also from UCI, [2] contains motion stream data of Australian sign language. For each of the 95 Australian language signs, 27 examples were captured from a native signer using high-quality position trackers and instrumented gloves.

Comparison Methods and Validation. For comparison, we implemented two basic classifiers: Nearest neighbor (1NN) and classification to the nearest class centroid (centroid). Both basic classifiers do not consider interaction patterns among time series. To classify a test object, for all training objects and all attributes the nearest neighbor classifier sums up the Euclidean distance and assigns the object to the class of that training object with the smallest overall distance. The centroid classifier computes for each class the centroid by averaging all time series-valued attributes of all training instances. A test object is assigned to the label of the closest centroid.

We obtained the code of TClass from the authors [94]. The parametrization of this method is very difficult, since the type of meta-features as well as the base learner need to be selected by the user. As recommended in [94], we use a universal set of meta-features for temporal domains. In addition, we use following aggregate global attributes: mean, min, max and mode. Mean,

[2]http://kdd.ics.uci.edu/databases/auslan2/auslan.data.html

max and min were recommended by authors for DS10. Mode improved results on DS1-DS8. In all experiments, we use AdaBoost with J48 decision tree as learner. According to the experiments in [94], this is the third-best base learner for TClass. Two slightly better learners are Naive Segmentation and voting. Although Naive Segmentation is implemented, we could not repeat the very good results. We do not use voting since it is not implemented and it is unclear, which voting algorithm was used in [94]. Because TClass is not deterministic, unless otherwise specified, we report the mean of accuracy of 10 runs.

In addition we compare our method to [145, 206, 146], 3 classifiers especially designed for EEG data (cf. Section 8.2). For DS1-DS4 train and test data are given. Classification results are therefore directly comparable. For DS5-DS8 we implement 5-fold cross-validation as in [206]. We do not provide a comparison to [204] since the authors arbitrarily selected runs from the full EEG data set. We validate DS9 with leave-one-out, since there are only few instances. For DS10 we implement 10-fold cross-validation as in [94].

Results. Our model-based classifier demonstrates an excellent accuracy of 100% on 6 out of 10 data sets and strongly outperforms the two basic classifiers. Table 8.2 summarizes the results. On EEG data, our classifier even outperforms the classifiers especially designed for this type of data [145, 206, 146]. These results demonstrate that the information on interaction patterns is highly relevant for the classification of EEG and fMRI data. On data set DS9 (fMRI) our model-based classifier shows very good results. Only one object is incorrectly classified. The most important models for classification decision, are judged as reasonable by domain experts.

Only on DS10, TClass performs significantly better than all comparison methods. But notice that DS10 has been originally published by the authors of [94] and TClass has been designed for motion stream data. With 75% in accuracy, our model-based classifier performs much better than the basic methods centroid and 1-NN. This indicates that attribute interactions are also useful for the classification of motion stream data. However, evidently, the interactions in this data set can not perfectly be captured by linear models. We could not reproduce the accuracy of TClass reported in the publication (98%) which was obtained using voting, since it is not clear which voting strategy has been applied. In addition, TClass with voting

Data Set	Centroid	1-NN	Models	TClass	Others
DS1	50%	50%	**100%**	70%	n.a.
DS2	55%	55%	**90%**	74%	n.a.
DS3	80%	55%	**100%**	72%	n.a.
DS4	60%	50%	**100%**	72%	96% [146]/ 98% [145]
DS5	65%	90%	**98%**	89%	n.a.
DS6	40%	80%	**100%**	86%	n.a.
DS7	90%	90%	**100%**	89%	n.a.
DS8	85%	90%	**100%**	91%	90% [206]
DS9	85%	77%	**96%**	58%	n.a.
DS10	11%	46%	75%	96%	**98%** [94]

Table 8.2: Results of Classification.

is very inefficient. The run[3] without voting for DS10 lasts approximately 24 hours. For 11 voters, as recommended in [94] a naive voting algorithm would take 11 days. The model-based classifier run needs less than 40 minutes under equal conditions. Moreover, our algorithm is parameter-free and deterministic, whereas TClass exhibits a very high variation of results. E.g. for DS1 accuracies between 20% and 85% are obtained.

As an additional advantage, the interaction models generated by our classifier provide interesting information for interpretation. Figure 8.3 displays the 3 most relevant models for separating the classes on DS1. For both classes, the layout of the 64 electrodes on the scalp is displayed together with the best separating interaction patterns. The models of the alcoholic subjects involve different brain regions than those of the healthy control group. The subtle differences of interactions need to be systematically evaluated in future work.

8.5 Conclusion

In this chapter, we proposed a novel model-based classifier for data with time series-valued attributes. Classification decisions are supported by class-specific interaction patterns within the time series of a data object. The experimental evaluation demonstrates that interaction patterns characterized

[3] 10-fold cross-validation; 33 runs; Intel Core 2 Quadro(2,66 GHz), 8 GB RAM

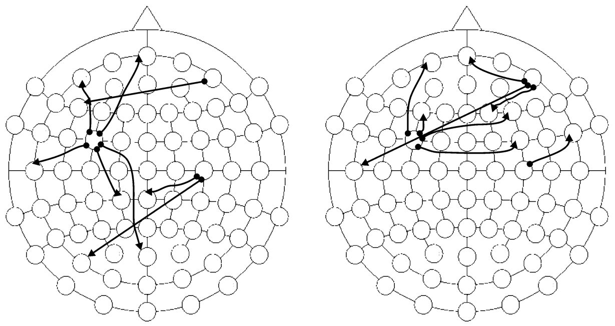

(a) Alcoholic. (b) Control.

Figure 8.3: Illustration of Best Class-separating Models on UCI EEG Data.

by linear models are very useful for classification, especially of EEG and fMRI data.

In ongoing work we focus on interpretation of the identified models. We will interpret the models of fMRI for somatoform pain disorder patients to find the most important regions. In addition, we will evaluate the classifier on EEG data from our cooperation partners for the identification of subjects in different stages of anesthesia. It would also be interesting to design model-based classifiers for fMRI data and combined fMRI-EEG data sets which is very challenging because of the large number of time series in fMRI data. We also plan to evaluate the classifier on time series data from other domains.

Furthermore, we want to extend our model notion to more general models. Among building blocks to be investigated are differential equations and kernel functions.

Chapter 9

Mining Interaction Patterns among Brain Regions by Clustering

Functional magnetic resonance imaging (fMRI) provides the potential to study brain function in a non-invasive way. Massive in volume and complex in terms of the information content, fMRI data requires effective and efficient Data Mining techniques. Recent results from neuroscience suggest a modular organization of the brain. To understand the complex interaction patterns among brain regions we propose a novel clustering technique. We model each subject as multivariate time series, where the single dimensions represent the fMRI signal at different anatomical regions. In contrast to previous approaches, we base our cluster notion on the interactions between the univariate time series within a data object. Our objective is to assign objects exhibiting a similar intrinsic interaction pattern to a common cluster. To formalize this idea, we define a cluster by a set of mathematical models describing the cluster-specific interaction patterns. Based on this novel cluster notion, we propose interaction K-means (IKM), an efficient algorithm for partitioning clustering. An extensive experimental evaluation on benchmark data demonstrates the effectiveness and efficiency of our approach. The results on two real fMRI studies demonstrate the potential of IKM to contribute to a better understanding of normal brain function and the alternations characteristic for psychiatric disorders.
The concepts described in this chapter have been published in Papers [154, 155] and extended with subsection 9.7.2. The paper [155] is an extended

version of the paper [154]. In this paper, Andrew Zherdin has proposed and optimized the application of the models for clustering. Besides, Andrew Zherdin has planned and carried out the experiments. Claudia Plant contributed the idea to apply the interaction-based model as a cluster notion and supervised the algorithm development. Anke Meyer-Baese has helped in the idea improvement. The medical part of the paper has been done by Christian Sorg and Afra Wohlschläger.

9.1 Introduction

Human brain activity is very complex and far from being fully understood. Many psychiatric disorders like Schizophrenia and Somatoform Pain Disorder can so far neither be identified by biomarkers, nor by physiological or histological abnormalities of the brain. Aberrant brain activity often is the only resource to understand psychiatric disorders. Functional magnetic resonance imaging (fMRI) opens up the opportunity to study human brain function in a non-invasive way. The basic signal of fMRI relies on the blood-oxygen-level-dependent (BOLD) effect, which allows indirectly imaging brain activity by changes in the blood flow related to the energy consumption of brain cells. In a typical fMRI experiment, the subject performs some cognitive task while in the scanner. Recently, resting-state fMRI has attracted considerable attention in the neuroscience community [53]. Surprisingly, only about 5% of the energy consumption of the human brain can be explained by the task-related activity. Many essential brain functions, e.g. long-term memory are largely happening during rest, most of them without consciousness of the subject and many of them are still not well understood. Therefore recent findings support the potential of resting-state fMRI to explore the brain function in healthy subjects and reveal alternations characteristic for psychiatric disorders (e.g. [179]) . In resting state fMRI, subjects are instructed to just close their eyes and relax while in the scanner. fMRI data are time series of 3-dimensional volume images of the brain. The data is traditionally analyzed within a mass-univariate framework essentially relying on classical inferential statistics, e.g. contained in the software package SPM [150]. A typical statistical analysis involves comparing groups of subjects or different experimental conditions based on univariate statistical tests on the level of the single 3-d pixels called voxels. Data from fMRI experiments are massive in volume with more than hundred thousands of voxels and hundreds of time points. Since these data represent complex brain activity, also the information content can be expected to be highly complex. Only a small part of this information is accessible by univariate statistics. To make more of the potentially available information accessible, we need effective and efficient multivariate Data Mining methods.

Recent findings suggest a modular organization of the brain into different functional modules [177]. To obtain a better understanding of complex brain activity, it is essential to understand the complex interplay among brain regions during task and at rest. Inspired by this idea, we propose a novel technique for mining the different interaction patterns in healthy and diseased subjects by clustering. At the core of our method is a novel cluster notion: A cluster is defined as a set of subjects sharing a similar interaction pattern among their brain regions. After standard pre-processing including parcellation into anatomical regions, we model each subject as a data object which is represented by a multivariate time series. Each of the dimensions is a time series corresponding to the fMRI signal of a specific anatomical brain region. Our approach Interaction K-means (IKM) simultaneously clusters the data and discovers the relevant cluster-specific interaction patterns. The algorithm IKM is a general technique for clustering multivariate time series and not limited to fMRI data. Besides fMRI, multivariate time series are prevalent in many other applications. Increasing amounts of motion stream data are collected in multimedia applications [34]. Gesture sensing devices, such as a CyberGlove usually contain multiple sensors to capture human movements. Human motion stream data can also be extracted from video streams. In this application, it makes sense to regard each movement as a data object. A cluster analysis of motion stream data potentially identifies clusters with similar movements, usually performed by different persons.

Clustering time series has already reached high maturity with multiple books and book chapters [147], surveys [114, 100] and a huge volume of research papers, e.g., [117, 196, 143, 142, 118, 162, 56, 199]. Defining a meaningful similarity measure is probably the most important challenge in clustering time series. In [116] Lin and Keogh illustrate some pitfalls associated with clustering based on subsequences with Euclidean distance and propose an alternative similarity measure based on characteristic recurrent motifs.

Considering the practical relevance of clustering multivariate time series, only disproportionately few papers address this issue [199, 202, 144, 200]. Many of the univariate methods mentioned so far can be straightforwardly extended to the multivariate case. However, by doing so, information is lost: Data which is inherently multivariate often contains interactions between the different time series. We demonstrate that this information can be very useful for clustering. In our examples, this aspect is intuitively reasonable: A motion is characterized by a specific pattern of dependencies among the recording sensors. A diseased person has a characteristic interaction pattern of brain regions which differs from the healthy controls. In [154] we first introduced this novel cluster notion. In this chapter, we extend this basic idea to support nonlinear models and demonstrate its potential to discover interaction patterns among brain regions from fMRI data.

The major contributions of this chapter can be summarized as follows:

1. We introduce a novel cluster notion for clustering multivariate time series based on attribute interactions.

2. We propose Interaction K-means (IKM), a partitioning clustering algorithm suitable to detect clusters of objects with similar interaction patterns.

3. We demonstrate that the information on interaction patters provides valuable insights for interpretation.

4. Motivated by a real challenge from a neuroscience application, IKM outperforms state-of-the art techniques for clustering multivariate time series on synthetic data as well as on benchmark data sets from different applications.

5. On fMRI data from studies on Somatoform Pain Disorder and Schizophrenia our algorithm detects very interesting and meaningful interaction patterns.

Notations. We consider a data set DS with n objects. Each object $O \in DS$ is a d-dimensional multivariate time series. Each dimension or attribute $a_i \in \mathcal{A}$ is a time series with m time points, i.e. $a_i = \{t_{1,i}, ..., t_{m,i}\}$. We also use $\vec{a_i}$ to denote the m time points of dimension a_i as a column vector. We use italics to denote sets, e.g. $\mathcal{A}$ denotes the set of attributes of DS and $\mathcal{O}$ a set of objects. Capital letters denote matrices composed by column vectors of dimensions. We further denote by m^* the overall number of time points considering one distinct dimension of some fixed set of objects. We consider a clustering $\mathcal{C}$ as a non-overlapping partitioning of DS into K clusters, i.e.

$DS = \bigcup_{1 \leq j \leq K} C_j$ and drop the indices whenever non ambiguous.
The remainder of this chapter is organized as follows. In the next section, we briefly survey related work. In Section 9.3 we introduce the interaction-based cluster notion. Section 9.4 extends the cluster model to support nonlinear interactions. In Section 9.5 we propose the corresponding clustering algorithm Interaction K-means (IKM). Section 9.7 contains an extensive experimental evaluation with diverse datasets and Section 9.8 shows solutions for neurological questions. Section 9.9 concludes the chapter.

9.2 Related Work

Time series clustering is nowadays an important and quickly developing field with application in medicine, astronomy and economics. In spite of importance of multivariate time series, only a few algorithms ([199, 202, 144]; [200]) have been especially designed for their clustering. However, most methods for clustering univariate time series can be applied to multivariate time series as well. Therefore we provide a brief survey on these techniques.

The most difficult task in clustering time series is to find an appropriate similarity measure. Many approaches rely on feature transformation and dimensionality reduction. Features derived by Discrete Wavelet Transform and the Discrete Fourier Transform [134], as well as obtained by Principle Component Analysis [182, 189] have been successfully applied for clustering. Alternative approaches to feature extraction include e.g. the method of multi-resolution piecewise aggregate approximation presented in [118]. This approach extends the idea of piecewise aggregate approximation to multiple resolutions. Recently, compression-based similarity measures have been proposed, e.g. [162]. A special representation of time series in a form of a bit string is used in this approach. These bit strings indicate the intervals where the value of the time series is above its mean value. To reduce the I/O cost and to speed-up the clustering task, these bit level representations are then compressed. A compression-based similarity measure (also proposed in [102]) is applied to compare long time series structure using co-compressibility as a dissimilarity measure. In spite of requiring certain statistical conditions from data, this is a good-working approach.

Many approaches use Hidden Markov Models (HMM) for clustering time series, e.g. [144, 161, 56], to mention a few. The concept of HMM is particularly useful to model temporal correlations among the measurements. The approach [56] for example uses segmental semi-Markov models. First, the time series are modeled by k distinct segments and then a Viterbi-like algorithm is applied to compute the similarity measure. In the experimental section, we compare to the Sequence Cluster Refinement Algorithm (SCRA) [202]. Like IKM, this approach is an iterative partitioning K-means-style clustering method but it uses HMMs to represent the cluster centers. In the assignment step, each time series object O is assigned to that cluster with the HMM model that most likely generated the sequence of O. In the update step, the HMM of each cluster is recomputed maximizing the probability of producing all time series currently assigned to the cluster. In contrast to HMMs, our models capture dependencies not between time points but between the dimensions. Our models are applicable to groups of objects and not to single objects. HMMs are also a good approach to features extraction from each dimensions of multivariate time series. So our models have a very different structure compared to HMMs. To describe the interaction patterns within the time series of an object, we apply linear models as recently proposed in [106, 24] for the purpose of efficient compression and classification of time series, respectively.

Some approaches especially focus on clustering multivariate time series. Authors of [200] suggest merging all dimensions of multivariate time series into a long univariate time and supplying it to the Autoclass algorithm [39]. This technique is not applicable if the number of time points varies among the objects, a case frequently occurring in fMRI data. In [202] authors apply Independent Component Analysis to transform multivariate time series data into statistically independent components, and then propose a clustering algorithm called ICACLUS to group underlying data series according to the ICs found. This approaches is based on the assumption that the observed multivariate time series are mixtures of statistically independent sources which need to be de-mixed in order to find the clusters. In the experimental section, we compare to ICACLUS, since Independent Component Analysis and related blind signal source separation techniques have been successfully applied for analyzing neuroimaging data, see e.g, [129]. We further compare our method to structure-based statistical features clustering (SF) [199] which has been especially designed for clustering multivariate time series. SF represents each attribute of the multivariate time series by a fixed-length vector whose components are statistical features of the time series which capture the global structure. Statistical features include e.g. hurst, kurtosis and nonlinearity.

These descriptive vectors, one for each component of the multivariate time series, are concatenated, before being clustered using a standard clustering algorithm such as K-means. Extensive experiments demonstrate that SF is suitable to identify activity patterns from motion stream data with high accuracy.

9.3 Interaction-Based Cluster Notion

In this section, we elaborate our cluster notion based on characteristic interaction patterns. We want to find clusters of objects which are represented by multivariate time series sharing a common cluster-specific interaction pattern among the dimensions. For an example consider Figure 9.1 displaying a data object represented by a multivariate time series. The dimensions of this object exhibit a simple interaction pattern: The time series of dimension dim_{12} can be expressed by a linear combination of some other dimensions: $dim_{12} := 2 \cdot dim_4 + dim_5 + dim_6$. Typically, not all dimensions of an object are interacting. For simplicity, only the dimensions involved in the interaction pattern are displayed Figure 9.1. Note that this is an overly simple example, but more complex patterns are prevalent in real-world data, and to the best of our knowledge, have not been exploited for clustering so far. Often, dependencies among dimensions are rather regarded as non-necessary redundant information which should be removed before clustering, e.g. by the application of PCA for linear dependencies. In this work, we want to preserve the information on attribute dependencies and demonstrate that it can indeed be very valuable for clustering. The resulting clusters are composed of objects exhibiting similar dependencies among their attributes which can therefore be interpreted as cluster-specific interaction patterns.

To formally represent this idea, we define each cluster by a system of mathematical models. We first focus on linear models. Linear models have been successfully applied in a wide range of applications because they are interpretable and computationally efficient. In Section 9.4, we consider non-linear dependencies, which can often be expressed by linearizable functions.

Definition 13 (Linear Model) *A linear model M_a for a dimension a and a set of objects $\mathcal{O}$ is provided by:*

$$A = V \cdot P + E, \text{ where}$$

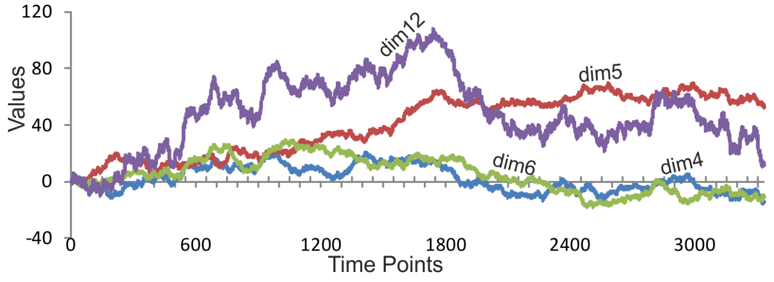


Figure 9.1: Example of an Interaction Pattern.

- *$A \in \mathbb{R}^{m^*}$ results from horizontally concatenating the time series of dimension a of all objects in $\mathcal{O}$,*
- *$V \in \mathbb{R}^{m^* \times |\mathcal{V}|}$ contains the explanatory variables,*
- *$P \in \mathbb{R}^{|\mathcal{V}|}$ contains the parameters or coefficients of the model.*
- *and $E = \sqrt{||A - VP||^2} \in \mathbb{R}^{m^*}$ contains the error.*

We further denote sum of errors of the set of objects $\mathcal{O}$ with respect to M_a by $\mathcal{E}_{M_a} = \sum_{i=1}^{m^*} E_i$. We can also determine the error of an arbitrary object $P \notin \mathcal{O}$ as $\mathcal{E}_{P,M_a} = \sqrt{||A_P - V_P P||^2}$. A_P denotes the matrix obtained by projecting object P to dimension a. Analogously, V_P is the matrix containing the set of explanatory variables to model object P.

To obtain a single model for each dimension of a set of objects, we concatenate the time series of all objects. Note that our method does not require that all time series of all objects are of equal length. To illustrate this definition, consider the model for dimension dim_{12} in Figure 9.1 as an example : Assume we have 100 objects in a cluster. In this case, A is of size $333,300$, since the time series of each object consists of 3333 time points. The explanatory variables for dim_{12} are dim_4, dim_5 and dim_6. Therefore, matrix V is of size $333,300 \times 3$. Now, we are ready to define an interaction-based cluster.

Definition 14 (Interaction-Based Cluster) *An interaction-based cluster C is defined by:*

- *A set of models $\mathcal{M}_C = \{M_{C,1}, ..., M_{C,d}\}$ representing the dependencies of each single dimension with respect to the other dimensions. We* *denote the model of dimension a_i of cluster C by $M_{C,i}$. For model $M_{C,i}$ we apply $\mathcal{V}_i \subsetneqq \{\mathcal{A} \setminus a_i\}$ as set of explanatory variables in Definition 13. We will address the question how to find a suitable set of explanatory variables below.*

- *A set of data objects $\mathcal{O}_C$ associated to C.*
- *We denote the error of cluster C by $\mathcal{E}_C$ which is provided by $\mathcal{E}_C = \sum_{i=1}^{d} \mathcal{E}_{M_i}$.*
- *The error of object O with respect to cluster C, denoted by $\mathcal{E}_{O,C}$ is provided by $\mathcal{E}_{O,C} = \sum_{i=1}^{d} \mathcal{E}_{O,M_i}$.*

The aim of interaction-based clustering is to obtain a non-overlapping partitioning of DS into K clusters. Finally, each cluster should represent a specific interaction pattern which is characteristic for the assigned objects. More precisely, the optimization goal can be specified as follows:

Definition 15 (Clustering Objective Function) *The optimization goal for interaction-based clustering is to minimize the total sum of errors of all K clusters.*

$$\min \sum_{C \in \mathcal{C}} \frac{1}{m_C^*} \mathcal{E}_C.$$

To take into account that clusters consisting of objects with longer time series naturally tend to have a larger error, we normalize $\mathcal{E}_C$ by the total number of time points of the objects in cluster C , which we denote by m_C^*.

Model Finding. Before addressing the problem of how to find the clusters, we need to describe how the set of models $\mathcal{M}_C$ can be computed from the set of objects $\mathcal{O}_C$ which are associated to a cluster C. Since we focus on linear models, this involves solving d regression problems. Multiple least square regression can be applied to derive the models. However, a common problem is overfitting. The more dimensions are included into the model, the more variance is explained and thus the smaller is the error term. Models involving a large part or even all dimensions are not generalizable and hard to interpret. Therefore, the set of explanatory variables of each model needs to be carefully selected.

To determine the really relevant dimensions, we apply a greedy-stepwise algorithm for model finding [110] in combination with the Bayesian Information Criterion (BIC) [57] as evaluation criterion. The greedy-stepwise algorithm is an established technique for variable selection in regression problems. This algorithm starts with an empty set of relevant dimensions. In each step, either one dimension is added or removed, depending on which of these two actions is judged superior by the evaluation criterion. The algorithm terminates if none of the two actions leads to a further improvement. As evaluation criterion, we apply BIC which determines a balance between goodness-of-fit and complexity of the model and is defined by:

$$BIC(M_a) = -2 \cdot LL(a, M_a) + \log(m^*)(|\mathcal{V}| + 1).$$

The first term represents the goodness-of-fit, where $LL(a, M_a)$ denotes the log-likelihood of dimension a given the model. The second term punishes overly complex models.

9.4 Nonlinear Models

Interaction processes in nature are not limited to be linear. Therefore we extend our approach to also support nonlinear models.

Definition 16 (Nonlinear Model) *A nonlinear model M_a for a dimension a and a set of objects $\mathcal{O}$ is provided by:*

$$A = f(V, P) + E, \ \textit{where}$$

- *A, V, P specified as in Definition 13,*
- *$E = \sqrt{||A - f(V, P)||^2} \in \mathbb{R}^{m^*}$ contains the error;*
- *Π is the number of parameters for the function $f(\cdot, \cdot)$*
- *and $f(\cdot, \cdot)$ is an explanatory function: $f : \mathbb{R}^{m^* \times |\mathcal{V}|} \times \mathbb{R}^{\Pi} \to \mathbb{R}^{m^*}$.*

The definition is analog to Definition 13 but extends it for nonlinear cases. If $f(\cdot, \cdot)$ is a classical multiplication of the matrix and a vector and $\Pi = |V|$, Definition 14 coincides with Definition 13.

Not every nonlinear model can be computed efficiently. We therefore consider only the sub-class of linearizable models. In linearizable models, each coefficient can be pre-computed and inserted as a new explanatory variable.

Definition 17 (Linearizable models) *A nonlinear model M_a for a dimension a and a set of objects $\mathcal{O}$ is provided by:*

$$A = \sum_{i=1}^{m'} f_i(v_i, p_i) + E, \text{ where}$$

- *A, V, P, E are specified analogously to Definition 13,*
- *m' is a number of summands,*
- *and $f_i(\cdot, \cdot)$ are a explanatory functions: $f_i : \mathbb{R}^{m^*} \times \mathbb{R} \to \mathbb{R}^{m^*}$*

A large class of nonlinear models can be specified by linearizable models. To give a simple example, consider the following:

$$A(t) = 3v_1^2(t) + 2\sin(v_2(t)) - log(v_3(t))$$

We can rewrite this model as:

$$A(t) = 3\widetilde{v}_1(t) + 2\widetilde{v}_2(t) - \widetilde{v}_3(t)$$

with $\widetilde{v}_1(t) = v_1^2(t)$; $\widetilde{v}_2(t) = \sin(v_2(t))$; $\widetilde{v}_3(t) = log(v_3(t))$.

Due to the special properties of fMRI data, models with time-delay are especially useful. An event in a specific brain region measured by BOLD signal activity induces BOLD activity in another region by some time delay [52]. Particularly Granger causality between time series of BOLD activity is well suitable for modeling causal (i.e. time delayed) influences of activity across remote brain areas, see e.g., [71]. To support integrating this aspect into our approach, we need a linearizable model with time delay.

Definition 18 (Model with time delay) *A model with time delay M_a is nonlinear model with the explanatory function f provided by:*

$$f : \mathbb{R}^{m^* \times |\mathcal{V}|} \times \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{m^* - |\mathcal{O}|}$$

$$f(V, P) := TimeShift_1(V) \cdot P \text{ where}$$

- *V, P are specified analogously to Definition 13*
- *and $TimeShift_1(V(t)) := V(t-1)$ i.e. we cut off the last time point of each object from matrix V. The length of output time series is $m^* - |\mathcal{O}|$, where m is the length of input time series (here $|\mathcal{O}|$ is the number of objects in the model).*

We can describe time-delay model for one object with a following equation:

$$TimeShift_{-1}(A) = TimeShift_1(V)P + E \text{ or}$$

$$A(t) = V(t-1)P + E(t) \text{ for } \forall t \in [2 \ldots m^*].$$

Our experiments in Section 9.8.3 demonstrate that models with time delay can improve the clustering accuracy on fMRI data. Depending on the application domain, other model classes like e.g. threshold autoregressive models for financial applications could also be integrated.

9.5 Interaction K-means Clustering

In this section, we introduce the algorithm interaction-K-means (IKM) which minimizes the clustering objective function provided in Definition 15. Similar to classical K-means [122], IKM is an iterative algorithm which efficiently converges towards a local minimum of the optimization space.

Algorithm IKM. Analogously to K-means, the first step of IKM is the *initialization*. As K-means, the result of IKM strongly depends on a suitable initialization. As a common strategy for K-means, we propose to run IKM several times with different random initializations and keep the best overall result. For initialization, we randomly partition DS into K clusters. For IKM it is favorable to assure that the initial clusters are balanced in size to avoid overfitting. Therefore, we partition the data set into K equally sized random clusters and find a set of models for each cluster as described in the previous section. After initialization, IKM iteratively performs the following two steps until convergence: In the *assignment* step, each object O is assigned to the cluster w.r.t. which the error is minimal, i.e. $O.cid = \min_{C \in \mathcal{C}} \mathcal{E}_{O,C}$. It is easy to see that this minimizes the objective function in Definition 15. After

```
algorithm IKM (data set $DS$, integer K): Clustering $\mathcal{C}$
  Clustering $bestClustering$;
//initialization
  for $init := 1 \dots maxInit$ do
    $\mathcal{C} := randomInit(DS, K)$;
    for each $C \in \mathcal{C}$ do
      $\mathcal{M}_C := findModel(C)$;

    while not converged or iter < maxIter do
//assignment
      for each $O \in DS$ do
        $O.cid = \min_{C \in \mathcal{C}} \mathcal{E}_{O,C}$
//update
      for each $C \in \mathcal{C}$ do
        $\mathcal{M}_C := findModel(C)$;
       if improvement of objective function
        $bestClustering := \mathcal{C}$;
    end while

  end for
return $bestClustering$;
```

Figure 9.2: Algorithm Interaction K-means.

assignment, in the *update* step, the models of all clusters are reformulated. Pseudocode of IKM is provided in Figure 9.2.

As an iterative partitioning clustering algorithm, IKM follows a similar algorithmic paradigm as K-means. However, note that there are significant differences: Our cluster notion requires a similarity measure, which is very different to LP metric distances. The similarity measure applied in IKM are the errors with respect to the set models of a cluster. This similarity measure is always evaluated between an object and a cluster, and not between two objects. In contrast to K-means or K-medoid algorithms, we can not state that a data object is the representative of a cluster. The cluster representative in IKM is a set of models describing a characteristic pattern of interaction among the dimensions.

Convergence. IKM converges as soon as no object changes its cluster assignment during two consecutive iterations. Usually, a fast convergence can be observed (less than 50 iterations on our experimental data), but there are some rare cases in which IKM does not converge. Analogously to standard K-means, it can be straightforwardly proven that the assignment and the update step strictly decrease the objective function provided in Definition 15. However, due to the greedy stepwise algorithm applied for model finding, cf. Section 9.3, the strictly monotonic property is lost. In particular, in different iterations of IKM, BIC may select different numbers of explanatory variables to be included in the cluster models. We therefore propose to terminate after *maxIter* iterations. Our experiments demonstrate that this has no negative effects on the quality of the clustering result. Recently, following the approach of smoothed analysis Arthur et al. [14] have proven that K-means converges in polynomial time on an arbitrary input data set subjected to random perturbations. This result theoretically supports our approach, since the fact that slightly different explanatory variable selection can happen in consecutive iterations is similar to minor perturbations of the data.

Runtime Complexity. As for ordinary K-means, the runtime of IKM scales linearly with the number of objects n, since the complexity of assignment step is linear in n and usually only a few iterations are performed. Clearly, the update step is the most computationally expensive step, since model finding involves matrix inversion with complexity of $O(d^3)$ combined with the greedy stepwise algorithm with complexity of $O(d^2)$. Aggregative pre-computing inspired by [40] allows us to become virtually independent of the length of the time series. For model-finding we very frequently need to determine the model parameters P and the error E, which involves matrix multiplication of complexity $O(m^*d^2)$. Following [40], we pre-compute the matrix Θ^* with $\theta^*_{ij} = \theta^*_{ji} := \sum_{k=1}^{m^*} Z_{ki} \cdot Z_{kj}$, $Z := [V|A]$. With Θ^*, multiplication can be performed very efficiently: $V^TV = \left(\theta^*_{ij}\right)_{\substack{i=1\ldots d-1,\\ j=1\ldots d-1}}$ and $V^TA = \left(\theta^*_{ij}\right)_{\substack{i=d,\\ j=1\ldots d-1}}$ $P = (V^TV)(V^TA)$ To determine the different models for the dimensions a cluster, we can always apply Θ^* and just need to apply another dimension as A. In contrast [40], we also need E which is outside the scope of [40]. But the computation of E can be supported by Θ^* as follows: We reduce $E^2 = ||A - VP||$ as follows:

$$\begin{aligned}
&E^2 = ||A - VP|| = (A - VP)^T(A - VP) = \\
&= A^TA - A^TVP - (VP)^TA + (VP)^T(VP) = \\
&= A^TA - (V^TA)^TP - P^T(V^TA) + P^T(V^TV)P,
\end{aligned}$$

with $(V^TA) \in \mathbb{R}^{(d-1)}$ and $(V^TV) \in \mathbb{R}^{(d-1)\times(d-1)}$, and $\mathbb{R} \ni (A^TA) = \theta_{d,d}$, cf. Definitions 13, 14. Calculation of Θ^* needs $O(m^*d^2)$, but only once. We calculate Θ^* as $\Theta^* = \sum_{i=1}^{|\mathcal{O}|} \Theta_i$. And Θ_i is precalculated for object i. $(\theta_i)_{kj} = \sum_{l=1}^{m} Z_{lk} \cdot Z_{lj}$. $|\mathcal{O}|$ is the number of objects in the cluster. The costs are amortized by each calculation of model coefficients and the error. Multiple runs of IKM increase this effect (cf. Section 9.7.3).

9.6 Interpretation of the Clustering Result

A major advantage of IKM is possibility to interpret the detected interaction patterns. To facilitate interpretation, we focus on a subset of the models which best differentiates among the clusters. For each pair of clusters, the best discriminating models are selected by leave-one-out validation using objects of the corresponding clusters. Figure 9.3 displays the algorithm for interpretation in pseudocode. Considering a pair of clusters, we first generate the models of each individual cluster of from the training data. Then we compute the error of the test objects w.r.t. all models and sum up all errors. To obtain a ranking of the models w.r.t. their ability to discriminate among the clusters, we consider errors w.r.t. the correct cluster of the test object with a positive sign (these errors should be small) and errors w.r.t the other cluster with a negative sign, respectively. Finally, we sort all models ascendingly according to the error. The top-ranked models best discriminate among the clusters.

User Feedback. The clustering result together with the information about which models best discriminate among clusters is a good basis for user interaction. Expert users can easily select the most relevant dimensions of the multivariate time series based on this information. Also, experts can easily verify their hypotheses on which dimensions, in the neuroscience application corresponding to anatomical brain regions are most relevant. After selecting the relevant regions, IKM can be run again. Our experiments in Section 9.8 demonstrate that this strategy can greatly improve the clustering result and thereby confirms hypotheses of the experts.

9.7 Comparison to State-of-the-Art

9.7.1 Methodology

Selection of Comparison Methods. Existing approaches to clustering multivariate time series do not consider interaction information. Most approaches rather extract features from each dimension and cluster the resulting feature vector, cf. Section 9.2. To demonstrate that the information on attribute interaction is valuable for clustering multivariate time series, we compare IKM to two feature-based approaches. As a baseline, we consider classical K-Means clustering with Euclidean distance, which we term *Naive* in the following. The naive algorithm considers the concatenated dimensions of an object as a feature vector. In addition, we compare to *Statistical Feature Clustering (SF)* [199] which is a state-of-the-art feature-based approach to clustering multivariate time series, cf. Section 9.2. We compare to two further approaches addressing the challenge of clustering multi-variate time series in different non-feature-based ways: The *Sequence Cluster Refinement Algorithm (SCRA)* [202] uses Hidden Markov Models (HMMs) to represent clusters. The technique *ICACLUS* [144] relies on Independent Component Analysis (ICA), see Section 9.2.

```
algorithm dimensionRanking
(Cluster $C_i$, Cluster $C_j$): ranking
  $error\_in\_models := new\ ARRAY[d]$;
//leave-one-out-validation
  for each $O \in \mathcal{O}_{C_i} \cup \mathcal{O}_{C_j}$ do
     $test := O$
     $\mathcal{O}_{C_i} := \mathcal{O}_{C_i} \setminus test$; $\mathcal{O}_{C_j} := \mathcal{O}_{C_j} \setminus test$;
     $findModel(C_i)$; $findModel(C_j)$
    for each cluster $\in \{C_i, C_j\}$ do
      if $O.cid = cluster.id$ then
        $sign := 1$;
      else
        $sign := -1$;
      end if
      for  $i := 1 \ldots d$ do
        $error\_in\_models[d] += sign*$
              $cluster.models[d].$
              $calculateErrorFrom(O.$
                    $getTimeSeries(d))$;
      end for
    end for
  end for
  sort($error\_in\_models$);
return  $error\_in\_models$;
```

Figure 9.3: Algorithm for Interpretation of the Results.

Implementation, Parametrization and Data Sets. We implemented our method, the naive approach and ICACLUS in Java. We implemented SCRA in Matlab and obtained the R-code of SF from the authors. For ICACLUS, we used the FastIca algorithm [85] to perform ICA. For SCRA, we built the cluster models as super-models from dimension-wise classical HMMs[1] since the paper lacks an exact specification and we could not obtain the code from the authors.

All experiments were performed on a workstation equipped with a 2.66GHz CPU and 8GB RAM. In all experiments, we set K to the number of classes in the data set for all techniques except ICACLUS. ICACLUS does not require the number as clusters as input parameter but two other parameter k specifying a number of relevant ICs and t specifying a similarity threshold. In all experiments, these two parameters were set to 1. We tried a range of different choices which all lead to worse results with excessive numbers of clusters. In addition, we set the maximum number of iterations for IKM to 75 in all experiments. For the efficient K-means-style methods IKM and SF, we used 100 random initializations and report the quality of the best result according to the internal objective function of the algorithm. Experiments with SCRA could only be performed with a single initialization due to excessive runtime, since the algorithm need hours to days to process a single data set. Experiments have been performed on 6 synthetic and 4 real-world data sets from various domains, for a summary see Table 9.1.

9.7.2 Effectiveness

To evaluate effectiveness, we report three established measures for clustering quality: the Rand Index (RI) [68] which is based on counting pairs of objects in the same class and in the same cluster and pairs of objects in different classes and clusters, respectively. Cluster Purity (CP) as introduced in [199] is the ratio of objects of the majority class of a cluster with respect to the size of the majority class. This is averaged among all clusters. Since RI and CP favor clusterings with imbalanced cluster sizes, we additionally report the Information Criterion (IC) [44]. IC is defined as the empirical conditional entropy between class- and cluster labels. Intuitively, IC corresponds to the number of bit required to encode the class labels of all objects given the cluster labels. For RI and CP, a good clustering obtains a high score. For

[1]http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html

name	type	domain	#objects	#cl.	#dim	length	reference
DS1	syn.	n.a.	600	6	13	3333	n.a.
DS2	syn.	n.a.	600	6	12	2048	n.a.
DS3	syn.	n.a.	120 - 640	6	12	2048	n.a.
DS4	syn.	n.a.	120	6	12 - 29	2048	n.a.
DS5	real	medical: EEG	20	2	64	256	[203]
DS6	real	motion stream	100	10	25	70	[193]
DS7	real	sp. recognition	640	9	12	7-29	[108]
DS8	syn.	n.a.	120	6	4 - 80	2048	n.a.
DS9	syn.	n.a.	120	6	12	100 - 4000	n.a.
DS10	syn.	n.a.	60 - 4200	6	12	2048	n.a.
DS11	real	medical: fMRI	26	2	90	216 or 325	[65]
DS12	real	medical: fMRI	26	2	5	180, 240, 300	[178]

Table 9.1: Characteristics of Experimental Data Sets.

IC a small value indicates a good clustering.

Synthetic Data

Proof of concept. We generated a synthetic data set (DS1) with 600 objects and 13 dimensions where each dimension is a time series with 3,333 time points. This data set contains six clusters. Each cluster consists of 100 objects which share a common interaction pattern as illustrated in Figure 9.1. In this data set, for each cluster four dimensions are interacting. The interacting dimensions are perfectly correlated with $R^2 = 1$. The times series of 12 dimensions (among them the four interacting ones) have been generated according to the random walk model. The remaining dimension is uniformly distributed. As expected, IKM perfectly clusters this data set, with optimal scores of all quality measures)cf. Table 9.2) and outperforms all comparison methods by a large margin. The second best result in terms of IC is obtained by SF (1.48), however this result has a RI of only 0.41 and CP of only 18%. The naive approach shows the worst IC of 2.38 but is somewhat better than SF in RI and CP. SCRA performs worst in all quality measures.

To demonstrate that IKM is designed to exploit interaction patterns for clustering, we created data set DS2 analogously to DS1 but without inter-

data set	method	RI	IC	CP
DS1 Synthetic	IKM	**0.99**	**0.09**	**99%**
	SF	0.49	1.48	18%
	ICACLUS	0.77	2.54	10%
	SCRA	0.17	2.58	14%
	Naive	0.72	2.38	30%
DS2 Synthetic w/o Interaction	IKM	0.77	2.54	10%
	SF	**1**	**0**	**100%**
	ICACLUS	0.97	0.15	64%
	SCRA	0.61	1.66	29%
	Naive	**1**	**0**	**100%**
DS5 EEG	IKM	**1**	**0**	**100%**
	SF	0.61	0.69	75%
	ICACLUS	0.51	0.3	10%
	SCRA	0.47	1.0	50%
	Naive	0.49	0.95	60%
DS6 CAD	IKM	0.91	0.91	67%
	SF	0.92	0.95	73%
	ICACLUS	0.85	2.21	19%
	SCRA	0.88	1.21	61%
	Naive	**0.98**	**0.2**	**90%**
DS7 Japanese vowels	IKM	**0.88**	**1.21**	**61%**
	SF	0.79	2.36	33%
	ICACLUS	0.79	2.88	15%
	SCRA	0.11	3.13	10%
	Naive	0.83	2.04	40%

Table 9.2: Results on Benchmark Data.

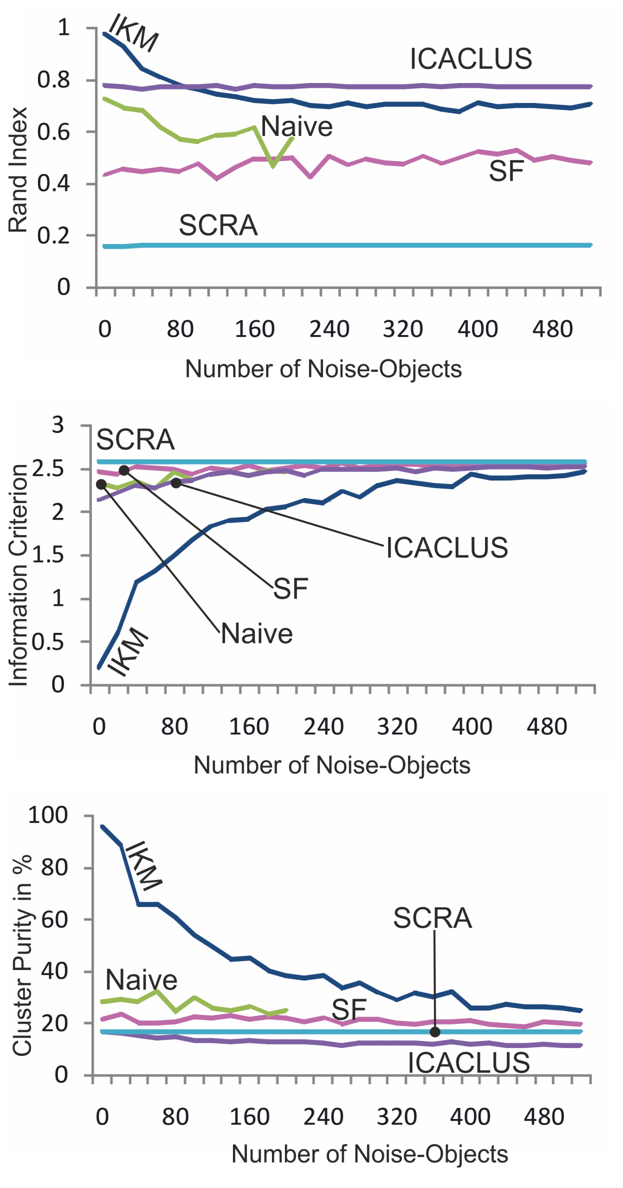
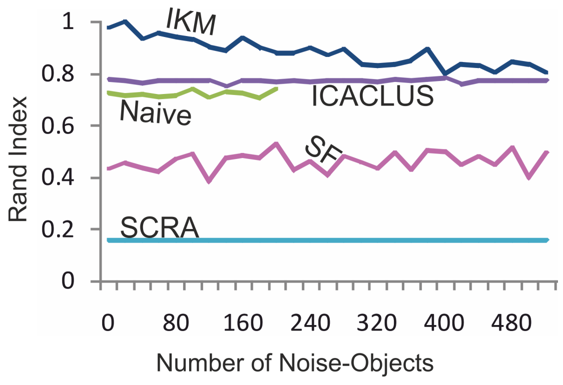
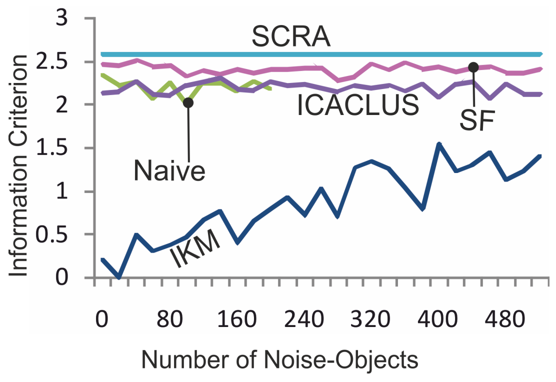


Figure 9.4: Impact of Noise Objects on Cluster Quality: Noise with Random Labels





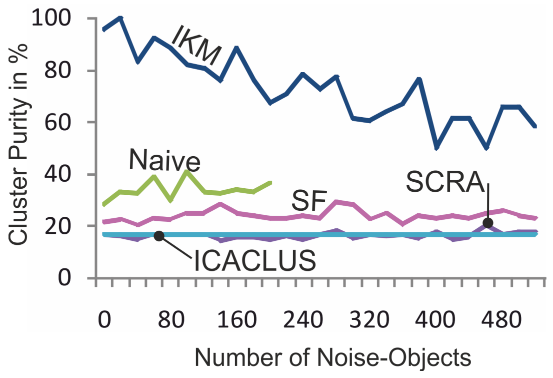


Figure 9.5: Impact of Noise Objects on Cluster Quality: Accuracy Without Noisy Objects

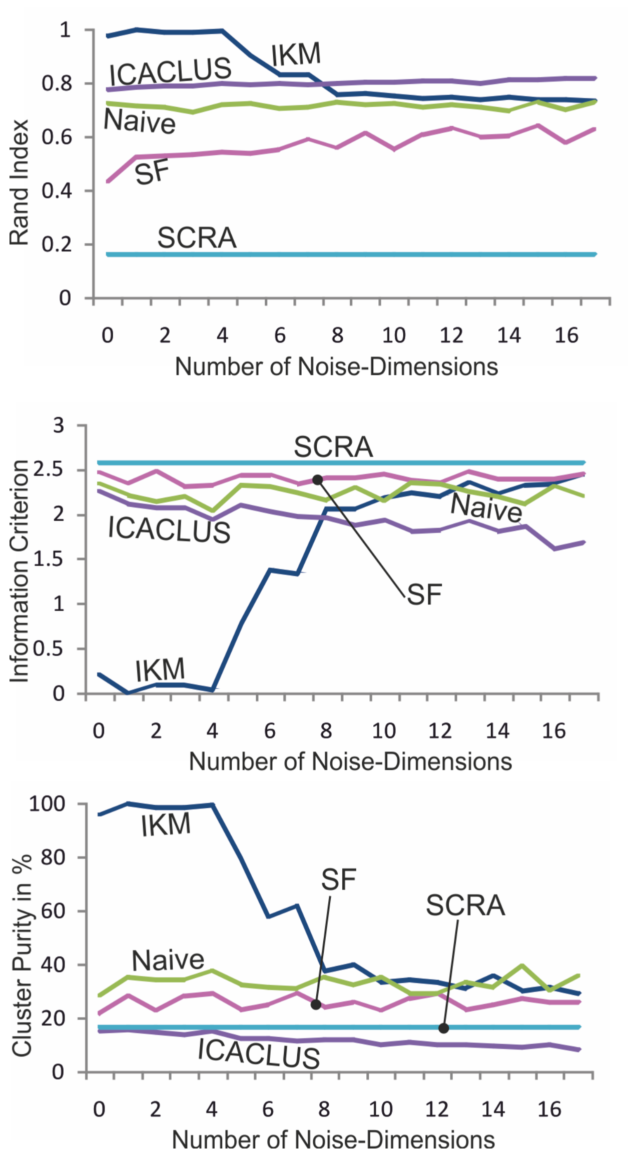
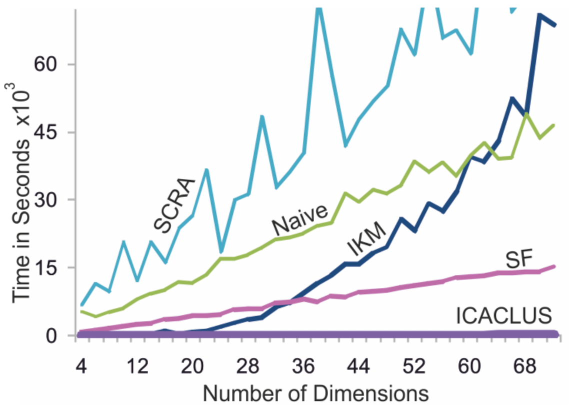


Figure 9.6: Impact of Noise Dimensions on Cluster Quality

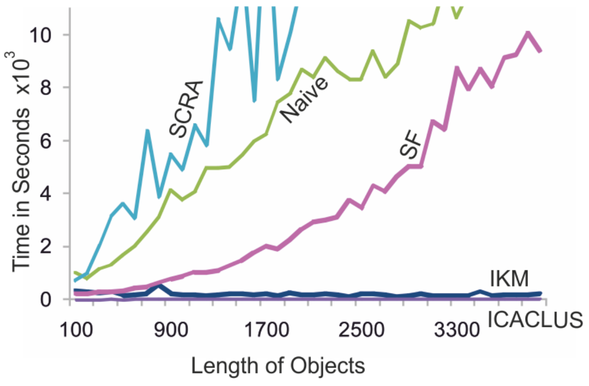
actions among the dimensions. Instead, following the assumption of blind signal source separation, we created the data of each cluster as a unique linear mixture of independent sources including sine and sawtooth functions. On this data set, IKM is outperformed by all comparison methods in terms of CP and IC.

Robustness against noise objects. Real data often contain a large amount of noise objects which often severely affect the result of clustering techniques. In the cluster model of IKM, a noise object is an object without any interaction pattern. To systematically evaluate the performance of IKM in the presence of noise objects, we created a synthetic data set (DS3) similar to DS1 (six clusters, 12 dimensions, length of time series: 2,048) with various amounts of noise. Starting with 120 objects in six clusters without noise we successively added in each step 20 noise objects up to 520 noise objects. For evaluation, we assigned random class labels to the noise objects and report the clustering quality measures for the whole data set (cluster objects and noise objects, cf. Figure 9.4)) and for the cluster objects only, cf. Figure 9.5. As expected, all clustering quality measures worsen with increasing amount of noise objects for IKM. For SF, Naive, ICACLUS and SCRA, the clustering quality stays constant at a very poor level. It is natural that the clustering quality for the overall data displayed in Figure 9.4 decreases when adding randomly labeled noisy objects. However, as Figure 9.5 demonstrates, IKM achieves a correct clustering of the cluster points even in the presence of large amounts of noise and always performs superior than SF and Naive. With 80% of noise, IKM obtains more than 50% of CP (Figure 9.5) or $RI \geq 0.8$ (Figure 9.5). At 82% of noise IKM is still better then the SF-method and Naive without noise. For some experiments (Figures 9.5 and 9.4), we did not launch the naive method on the very big noised datasets. On one hand this would take a several weeks of calculation and on the other hand the weak result was not going to improve with increase of noise.

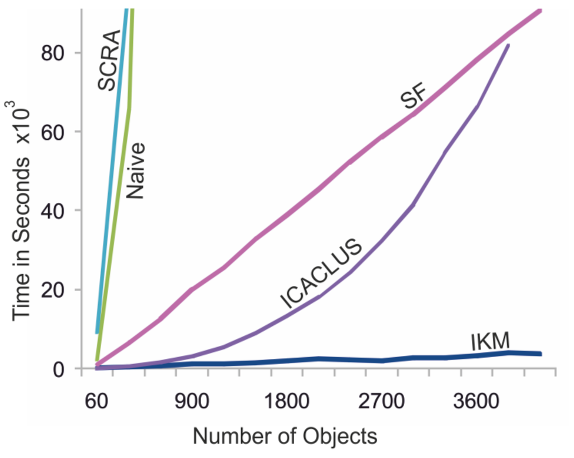
Robustness against noise dimensions. To study the robustness against noise dimensions, we successively added an increasing number of noise dimensions to our basic 12-dimensional data set (DS4). Noise dimensions have random values at all time points. Figure 9.6 demonstrates that up to four noise dimensions do not affect the clustering accuracy of IKM, which corresponds to 25% of noise. More than 10 noise dimensions affect the result of IKM, but this means a noise percentage of 40% and more. The accuracy of



(a) #dimensions/time



(b) length/time



(c) #objects/time

Figure 9.7: Scalability w.r.t. Dimensionality, Time Series Length and Number of Objects

the comparison methods remains constant, but at a pretty poor level. IKM outperforms them by a large margin up to 40% of noise and still remain better up to 60% of noise dimensions.

Real Benchmark Data

EEG data. Our first real-world data set DS5 is the EEG data set available at the UCI machine learning repository[2]. This data set is derived from a study on alternations of brain activity in alcoholic subjects in comparison to a control group. Each subject was exposed to either a single visual stimulus (S1) or to two visual stimuli (S1 and S2). When two stimuli were shown, they were presented in either a matched condition where S1 was identical to S2 or in a non-matched condition where S1 different from S2. Data set DS5 is derived from the so-called Small Data Set which contains data of 2 subjects, one alcoholic and one of class control. For each of the 3 experimental paradigms, 10 runs of EEG have been recorded. The task here is to cluster the EEG runs into alcoholic and control. IKM yields a perfect clustering with RI = 1, IC = 0 and CP of 100%. SF is the best performing comparison method with RI = 0.61, IC = 0.69 and CP of 75%. Besides the clustering, the result of IKM specifies the most important differences among the interaction patterns of the electrodes of the alcoholic subject and the healthy control. The most important electrodes for clustering are CZ (ch. 15), F8 (ch. 3), FT7 (ch. 36) and C3 (ch.16). The lateral and medial frontal brain areas under these electrodes can be associated with decision making processes. Substance-dependent individuals show disadvantageous decision-making, as well as altered frontocortical recruitment when performing experimental tasks [80, 49].

Common Activities data set. The data set DS6 from [193] consists of motion stream data of ten different activities performed by one person, for example pick up an object, wave or talk on a cell phone. There are ten different instances collected for each activity, hence 100 sequences in total. It is a non-trivial task to recognize the activities since they are typically performed with different rates of motion execution. Each activity is been represented by a multivariate times series with 25 dimensions and 70 time points after Kernel PCA preprocessing with an RBF kernel [167]. On this

[2]http://archive.ics.uci.edu/ml/databases/eeg/eeg.data.html

data set, SF performs slightly better in Rand Index and Cluster Purity than IKM (cf. Table 9.2). The results reported for SF in [199] are even superior (only CP between 84% and 89%). We could not reproduce these results. Probably, the authors applied a different RBF-kernel for kernel PCA, which is unfortunately not exactly specified in [199]. However, the overall best result is achieved by the naive algorithm which is with 90% cluster purity even better than the results of SF reported in [199]. This demonstrates that our kernel PCA successfully captures the cluster separating information. In summary, we conclude, that interaction-based clustering is also successful in the application of motion stream data, but does not outperform SF and Naive.

Japanese Vowels. The real-world data set DS7 consists of two Japanese vowels uttered by nine different male speakers and is available at the UCI machine learning repository [3]. The task is to recognize the speakers based on 12-dimensional time series consisting of the Cepstrum coefficients of the speech signals. Representing speech by Cepstrum coefficients is wide-spread in language processing and speech recognition [108]. Cepstrum coefficients model the time evolving signal as an ordered set of coefficients representing the signal spectral envelope. The special characteristics of this data set are that time series vary very much in length (between 7 and 29 time points). We merge training set and test set for clustering. The results (cf. Table 9.2) demonstrate that IKM is the best choice to cluster the speakers since it clearly outperforms all comparison methods.

Search for cluster number in datasets

One of the important k-means extensions is an ability to automatically determine the number of clusters in the dataset being analyzed. In the extensions for the classical algorithm, the distance between the objects is used for this, as for example Silhouette Coefficient in the paper [97]. Our method does not allow to measure a "distance" between two objects. We consider here the average object error in its own cluster model. This value should decrease very quickly until the optimal number of clusters k is reached. The further decreasing of the average error should be inessential.

We analyze six datasets, five of which are real-word and one is synthetic. Applied on some datasets, we get expected results, whereas k determination

[3]http://archive.ics.uci.edu/ml/datasets/Japanese+Vowels

is impossible when applied on the others.

In the picture we see that the right cluster number labeled with a red X coincides with the break of the curve. The error decreasing in this point decelerates essentially at further growing of the number of clusters k. Automatic search for this point does not represent a challenge and is not considered here. In the Figure 9.8(C) is presented a reduced DS11, as in Subsection 9.8.2 (after user interaction, see Table 9.3). Here it should be noted the excellent clustering quality of these datasets (see Tables 9.2 and 9.3).

In the Figure 9.9 it is almost not seen any slowing down in decreasing at the point marked with X. For these datasets, such a method of cluster number determination does not bring visible positive results. We think, that this can have to do with the absolute clustering quality. For example, DS1, DS5 and the reduced DS11 are partitioned almost ideally. Clusterings for DS6, DS7 and the original (full) DS11 does not entirely correspond to the object labels. Their models probably include more noise and the average errors in their own clusters dwindle the critical importance.

9.7.3 Efficiency

Table 9.1 provides a summary on the data sets DS8-DS10 used for evaluation of scalability. These synthetic data sets are generated similar to DS1 varying all important properties. DS8 has 120 objects with a dimensionality ranging from 4 to 80. DS9 has 120 objects of the length from 100 to 4,000 with 100 as increment. DS10 consists of different number of objects.

Figure 9.7(a) shows scalability in the dimensionality of the time series. For this example, the increase in runtime is approximately $O(d^3 \cdot \log(d))$ for IKM. ICACLUS, SF and Naive scale linearly in the dimensionality, where ICACLUS is most efficient on all data sets. SCRA is the least efficient method in comparison.

Figure 9.7(b) shows scalability for the length of the time series. Although the complexity is in principle linear for IKM, it can be very good amortized (we do not see time-growing). Similar to IKM, ICACLUS also scales very well. SF needs $O(d^2)$ of CPU-time. Naive is linear, but slower on our test data. Again, SCRA is least efficient. The high variation in runtime of SCRA is due to the fact that the algorithm depends very much on the initialization.

The Figure 9.7(c) shows scalability in number of objects. The complexity is linear for IKM, SF, Naive and SCRA. ICACLUS scales quadratic in the

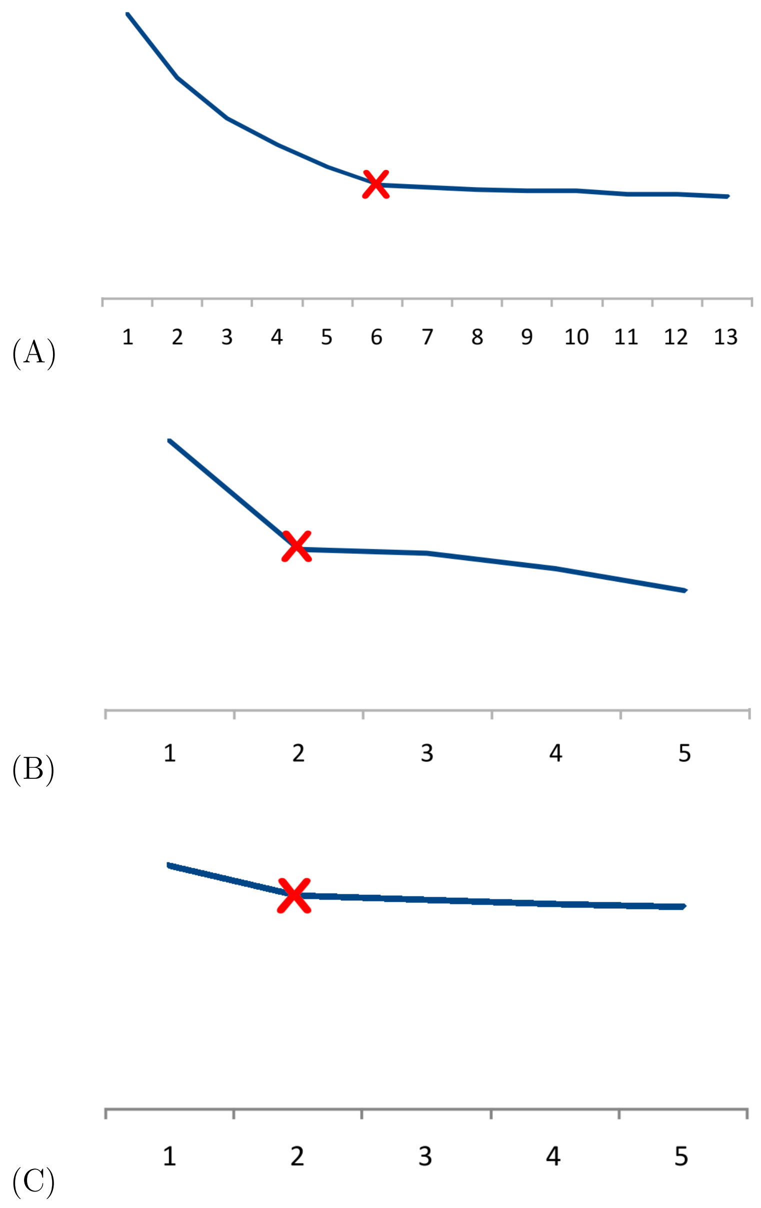


Figure 9.8: The average error of objects in their own cluster / Number of clusters in clustering (A) DS1, (B) DS5 and (C) reduced DS11.

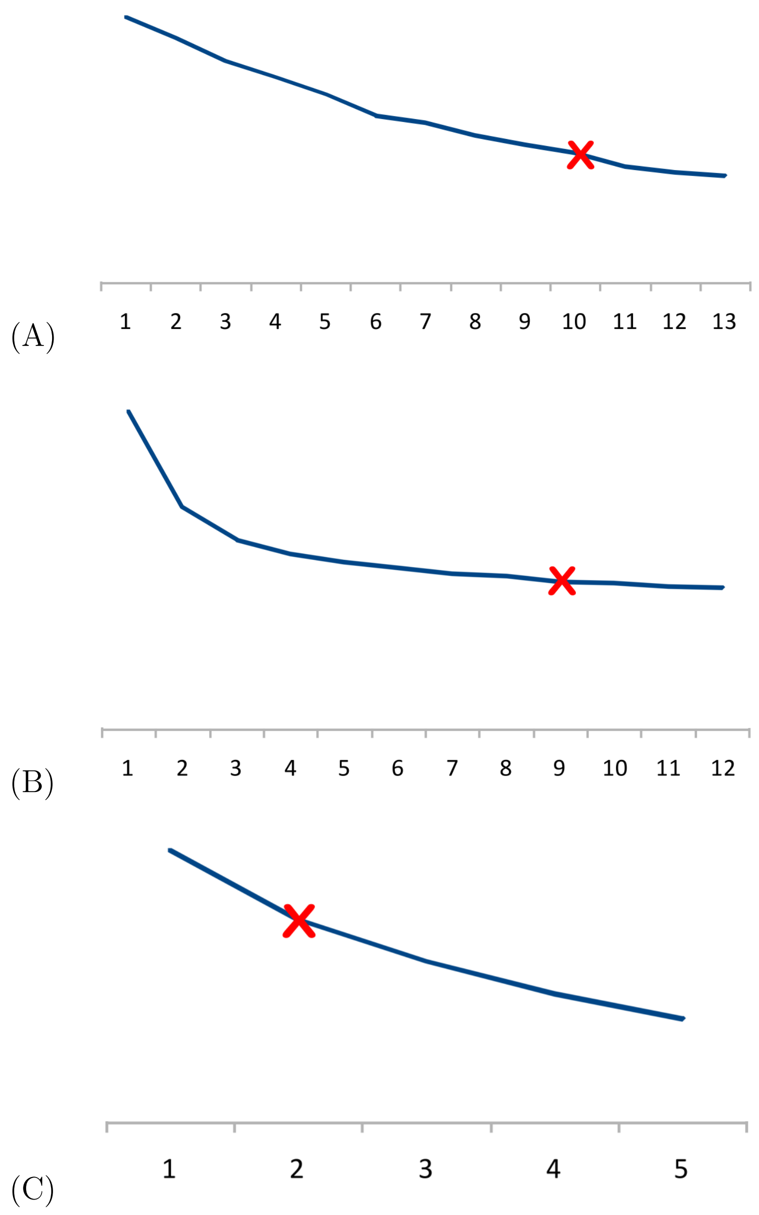


Figure 9.9: The average error of objects in their own cluster / Number of clusters in clustering (A) DS6, (B) DS7 and (C) DS11

number of objects. Nevertheless, the naive algorithm and SCRA are the slowest methods on all test data sets.

9.8 Interaction among Brain Regions

9.8.1 Functional Magnetic Resonance Imaging

We obtained data sets DS11 and DS12 from different functional MRI experiments. Functional MRI generates a series of 3-D volume images of the brain. Each image consists of about 60,000 voxels and the interval between time points is about 2-3 seconds. Functional MRI indirectly measures the strength of neuronal activity by the blood-oxygen-level-dependent (BOLD) effect. Brain regions with high neuronal activity consume larger quantities of oxygen-rich blood than inactive regions. Different physical properties of oxygenated and deoxygenated blood allow for indirect signalling of neuronal activity by MRI. We first applied standard pre-processing including realignment, normalization to a standard template and smoothing. Our approach is based of a set of time-series. Basically we can use each voxel time series from the images. However, for neighboring voxels signal activity is very similar. Moreover, medical experts often desire to obtain results at the level of anatomical regions which facilitates interpretation. Therefore, in Section 9.8.2, we use a brain atlas from [190] with a predefined mask of regions. As an alternative to anatomical regions, in Section 9.8.3 we use Independent Component Analysis (ICA). From the ICA result, we selected physiologically relevant components and rejected ICs reflecting motions-artifacts or noise.

9.8.2 Somatoform Pain Disorder

DS10 [65] has been obtained from a study on Somatoform Pain Disorder and consists of images of 13 subjects with pain disorder and 13 healthy controls. Somatoform Pain Disorder has severe impact on the quality of living of the affected persons since the main symptom is severe and prolonged pain for which there is no medical explanation. The causes of this psychiatric disorder are not fully understood but the hypothesis is that patients have altered mechanisms of observing and processing pain. Therefore, in our experiment, subjects underwent alternating blocks of pain- and non-painful stimulation while in the scanner. After pre-processing we segmented the data of each subject into 90 anatomical regions of interest [190] (ROIs). The task is to cluster persons based on the interaction patterns of the ROIs within the brain during the experiment. Each person is represented by a multivariate time series with 90 dimensions and 325 time points. There are four subjects with 216 time points only. Our technique IKM does not require the multivariate time series subjected to a cluster analysis to be of equal length. For Naive and ICACLUS we can only use 216 time points for clustering, which implies a considerable information loss. A cluster analysis with IKM provides valuable insights into disordered brain connectivity of patients with pain disorder. Table 9.3 shows the clustering results. The result of IKM is superior to the results of all comparison methods: One cluster is composed of nine subjects with somatoform pain disorder and four healthy controls. The second cluster contains nine healthy controls and four subjects with somatoform pain disorder. Based on previous studies [65, 181], it is known that the right amygdala is strongly associated with somatoform pain disorder. The model of this region is the best separating model among the clusters. Figure 9.10 presents a visualization of the model of right amygdala. The coefficients of this model are represented by color coding.

User Interaction. Based on the model displayed Figure 9.10, our medical experts refined the set of ROIs in our input data to four regions in the orbitofrontal cortex: Inferior orbitofrontal(right and left) cortex, medial orbitofrontal cortex (right and left). These regions are also known to be involved in the representation of subjective feelings including pain [107]. After user interaction, IKM obtains a nearly perfect clustering: (cf. Table 9.3): Only one patient is put in a wrong cluster. The comparison methods do not profit much from the dimensionality reduction performed by our experts. This demonstrates that the cluster model of IKM successfully represents interaction patterns among brain regions. Furthermore, IKM yields interpretable results which can be improved by user interaction.

9.8.3 Schizophrenia

The dataset DS12 [178] is derived from 26 persons including 13 healthy controls and 13 patients with schizophrenia, who all were assessed by 10-minutes of resting-state functional MRI. Schizophrenia is characterized by the impaired interaction between distributed brain regions particularly the striatum. Increased dopamin activity in the striatum is essential for schizophrenia

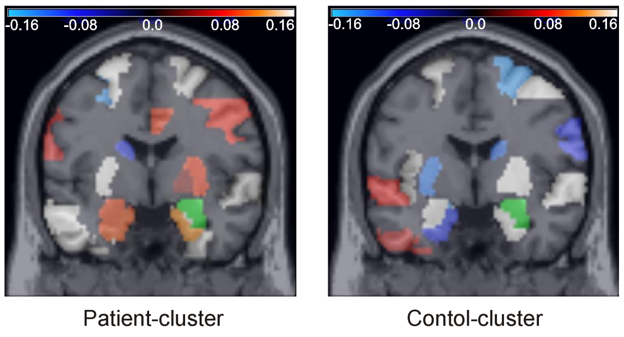


Figure 9.10: Graphic representation of the models for the right amygdala (green) in patients and controls involving all 90 AAL-ROIs. While healthy controls show functional connectivity of the amygdala to sensory areas (superior temporal, auditory), patients show increased connectivity to frontal control areas (superior frontal). Red to white: areas with a signal-time course with positive linear coefficient in the model, Blue: negative linear coefficient, respectively.

and antidopaminergic treatment the main therapy of the disorder. Therefore we suggested that the causal influence among intrinsic brain networks including the striatum is aberrant in patients. Intrinsic brain networks are characterized by synchronous brain activity at rest. (see Figure 9.11 for an example: spatial map of the intrinsic basal ganglia network including the striatum). Independent component analysis of fMRI data resulted in 9 ICs representing intrinsic brain networks by spatial maps of synchronous activity and corresponding time series. These time series result in 26 multidimensional time series objects of healthy controls and patients. Causal influence from one area into another was modeled by Granger Causality between time series of brain network activity.

Clustering based on nonlinear models reflecting Granger causality separated patients from controls with high cluster purity (84.6%) consistently for a model of the striatum. In each cluster of 13 persons were only 2 person assigned to the wrong cluster. Changed influence on the striatum was found for several intrinsic brain networks, indicating an aberrant regulation of striatal activity. These data suggest that altered regulatory intrinsic network activity contributes to increased striatal dopamin function.

data set	method	RI	IC	CP
DS11 Somatoform	IKM	**0.56**	**0.89**	**69%**
	SF	0.48	1	50%
	ICACLUS	0.53	0.18	9%
	SCRA	0.48	1	50%
	Naive	0.49	0.98	58%
DS11 Somatoform after user interaction	IKM	**0.92**	**0.20**	**96%**
	SF	0.48	0.99	54%
	ICACLUS	0.54	0.18	35%
	SCRA	0.48	1.0	50%
	Naive	0.48	1.0	50%
DS12 Schizophrenia	IKM-linear	0.52	0.92	65.38%
	IKM-nonlinear	**0.73**	**0.62**	**84.62%**
	SF	0.49	0.92	57.69%
	Naiv	0.48	1.00	53.85%
	ICACLUS	0.52	0.83	27.69%
	SCRA	0.48	1	50%

Table 9.3: Results on fMRI Data.

Table 9.3 shows the results of nonlinear and linear model-based IKM and the results of the comparison methods for DS12. The datasets consist of two clusters of patients and healthy controls, which are unambiguously and objectively determined. Nonlinear IKM clearly outperforms all comparison methods.

9.9 Conclusion

In this chapter, we propose a novel cluster notion for multivariate time series. We define a cluster as a set of objects sharing a specific interaction pattern among the dimensions. In addition, we propose interaction K-means (IKM), an efficient algorithm for interaction-based clustering. Our experimental evaluation demonstrates that the interaction-based cluster notion is a valuable complement to existing methods for clustering multivariate time series. IKM achieves good results on synthetic data and on real world data from various domains, but especially excellent results on EEG and fMRI

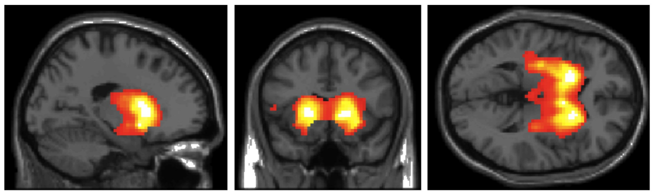


Figure 9.11: Spatial map of the intrinsic basal ganglia network including the striatum.

data. Our algorithm is scalable and robust against noise. Moreover, the interaction patterns detected by IKM are easy to interpret and can be visualized. Nonlinear models show their superiority in the corresponding real world data. In ongoing and future work, we plan to extend our ideas to differential equations. We want to consider different models for different regions of the time series. We intend to work on methods for suitable initialization of IKM, since existing strategies for K-means can not be straightforwardly transferred to IKM because of the special cluster notion. We are also investigating in feature selection for interaction-based clustering.

Part IV

Medical Applications

Chapter 10

Hierarchical Model-based Clustering of Neuroimaging Data

In this chapter, we consider various hierarchical clustering algorithms defined on mathematical models. Extensions of such classical approaches as Single- and Average-Link for models are proposed here. Also, we present two hierarchical subspace clustering algorithms for model-defined clusters.
These algorithms have shown excellent results on real fMRT measurements taken from Alzheimer's disease patients and healthy controls. Motion stream data[193] and japanese vowels dataset[108] are another examples of a successful usage of presented here approaches of hierarchical subspace clustering based on mathematical models.
The concepts described in this chapter are based on Paper [149] as well as on yet unpublished material. In this paper, Andrew Zherdin has proposed to use models for hierarchical clustering and implemented the algorithm for that. Besides, Andrew Zherdin has carried out the experiments. Claudia Plant and Annika Tonch have optimized my algorithm and carried out a part of the experiments. The other coauthors have developed the medical part of the paper.

10.1 Introduction

Many objects in the world around us are grouped not into a simple flat system of groups, but build hierarchical group structures. Especially in the medical applications, the relationships between objects and groups of objects help to find the disease reasons and describe its circuit. Here we compare the groups of patients with varying severity degrees and healthy controls of different ages. We are looking for the group, which is the closest to aged healthy controls. If the closest group is the initial disease stage of MCI, then the disease is rather a result of aging process. If the group of aged healthy controls is closer to the younger healthy controls, then the hypothesis about Alzheimer's disease is most likely false.

Having groups of younger (ca. 25y) and older healthy persons (ca. 70y), patients with MCI and AD dementia (ca. 70y), we tested the hypothesis that intrinsic activity of elder controls has more similarities with the intrinsic activity of patients than the same of younger persons. The assumption is conditioned by potential impact of preclinical AD. To check this, we used resting-state fMRI, various measures of intrinsic brain activity, and different hierarchical clustering methods. All these approaches confirm our assumption, that intrinsic activity among elder controls and patients are more alike than that of the younger controls. The linkage between these two factors in behavior of intrinsic activity, AD and age, may contribute critically to the pathogenesis of AD.

An important kind of clustering search is subspace clustering. For complex multidimensional objects, interesting dependencies often arise only in some subset of dimensions. Not taking into account such a phenomenon, it is very hard to find a reasonable clustering, since irrelevant dimensions (which could be very many of) bring a lot of noise to the overall picture. It can frequently be observed that with decreasing of dimensions subset, the clusters merge to bigger ones. Such a clustering is called subspace clustering. We can select certain dimensions we are interested in. By this we get a hyperplane projection. This approach we call Hierarchical Axis-Parallel Subspace Clustering. If we transform data space to a new one having a smaller number, then this method is called Hierarchical Subspace Clustering in Arbitrarily Oriented Subspaces.

We carry over the idea of hierarchical subspace clustering to model-based clustering approach. We consider two strategies of constructing hierarchical clustering in the variable dimension spaces and explore benefits and disadvantages of these methods. The results we get show the efficiency of the models also in searching for hierarchical structures on the subsets of dimensions.

10.2 Related Work

A classical example of the algorithms, searching for hierarchical dependencies as in the case of hierarchical clustering, are single- and average-link from book [90].

During initializing, each object is placed in a separate cluster. Further, it is supposed to measure the distance between all cluster pairs and merge two closest clusters in one. Essential is here the definition of distance between two clusters. This could be either minimal or maximal distance between objects of corresponding clusters (single-link or complete-link from book [90]). Alternatively, we can define the distance between two clusters as the average distance of all pairs of objects. Each such a pair consists of objects from both clusters (average link from the same book [90]).
The interpretation of the resulting hierarchical clustering provides us not only with information about the relationships between objects, but also about the relationships between groups of these objects. We try to find similar and distinct groups by means of hierarchical clustering.

Along with being the most spread cause of neurodegenerative disease, Alzheimer's disease (AD) is also the major reason of age-related dementia [33, 21]. Almost 50% of 90-years old persons suffer from AD dementia whereas at the age of 60 there are only 1-2% such people [79]. Among some other factors influencing the appearance of sporadic AD like genetic predispositions or lifestyle, the age is counted without doubt to the most important one [81]. It is hypothesized (the amyloid hypothesis), that β pathology (Aβ) is the critical initiating event in AD [171], which according to researches [17, 88], can start much earlier (up to 30 years before) than the first symptoms of AD appear. Linking aging, sporadic AD, Aβ-pathology and increased neuronal activity, a recent hypothesis suggested that lifespan intrinsic (i.e. ongoing) brain activity particularly in hetero-modal areas of high levels of functional connectivity may trigger regional Aβ-pathology and therefore the rise of AD[31, 89]. Given that AD may have a 20-30 years long preclinical period, this model suggests that due to advanced age and

therefore potential impact of preclinical AD, intrinsic activity of older persons resembles more that of patients than of younger controls. While most studies have investigated the impact of either aging or disease on brain's activity [62, 61, 58, 148], few studies have compared brain activity across healthy young, healthy elderly and patients with AD [32, 67, 183]. In particular with respect to intrinsic brain activity, a systematic approach focusing on the degree of similarity or dissimilarity across groups instead of simple group differences is - to the best of our knowledge - missing.
For hierarchical clustering of big and complex objects it is reasonably to use only a subset of dimensions. That is, to select only interesting for us dimensions or to transform the data into a new space with a less number of dimensions, without information loss about relations between objects and clusters. When choosing only interesting dimensions one speaks about Hierarchical Axis-Parallel Subspace Clustering. In the case of data transformation into a new space, where a hierarchical clustering then applied, one calls such algorithms Hierarchical Subspace Clustering in Arbitrarily Oriented Subspaces.

Axis-Parallel Subspace Clustering

CLIQUE (CLustering In QUEst) [8] represents a grid-based algorithm, where possible subspaces are recursively selected in a bottom-up way. The input parameter for the method are ξ and ρ, where ξ is the size of unit (block, the data space is partitioned into) and ρ is the density threshold, starting from which the concrete unit is considered as interesting. Following bottom-up technique, units for each next step (dimension) are generated via current step (dimension) units and self-join. Not dense candidates at each step are dropped. Then, clusters are found as a maximal set of connected dense units.

SUBCLU (density-connected SUBspace CLUstering) [95] is a bottom-up greedy algorithm which uses DBSCAN [46] cluster model of density-connected sets instead of grids. The input parameter for the method are ϵ and μ from DBSCAN approach. The algorithm starts with finding 1-dimensional clusters according to DBSCAN. Iteratively, we move from current dimension to the next one by checking if the clusters, we have so far, are also existent in one or more subspaces of higher dimension. DBSCAN is then applied on these candidates to generate the clusters for the next dimension. The algorithm stops when the set of subspaces containing clusters is empty. In comparison with grid based algorithms, SUBCLU gives a better clustering quality, however at the price of longer runtime.

PROCLUS(PROjected CLUStering) [5] represents a greedy k-means like algorithm. Its input parameters are the number k of clusters and the average subspace dimensionality l. Using greedy technique, potential medoids (at least one medoid per cluster) are selected. Starting from a choice of a certain set of medoids as cluster representatives (one medoid per cluster), these representatives are iteratively improved via replacing not suitable medoids with randomly chosen new ones. The measure of clustering quality is the mean distance between objects and their closest medoids. By means of special statistics, the relevant dimensions of cluster subspace are determined, after which so called Manhattan segmental distances are computed to assign objects to the cluster. Final improvements are done via recalculation of new subspaces and reassigning the objects to the clusters. PROCLUS method is known to be very sensitive to input parameters, especially to the average subspace dimensionality l.

Hierarchical Subspace Clustering in Arbitrarily Oriented Subspaces

ORCLUS (arbitrarily ORiented projected CLUSter generation) [6] represents an advanced version of PROCLUS method [5], finding correlation clusters in arbitrarily oriented subspaces. Same as in the case of PROCLUS, the number k of clusters and the average subspace dimensionality l are input parameters. In the first step of the algorithm, every data point is assigned to its nearest seed by means of the distance in the current subspace. This action divides the dataset into a certain number of current clusters, whose centroids become new seeds. In the next step, for each current cluster, by means of covariance matrix and eigenvectors, the algorithm determines the corresponding subspace. It is important that at each iteration, the current subspace dimensionality is reduced by some coefficient β. The final step is merging clusters being close to each other and having similar directions. The number of clusters is thereby reduced by some factor α. The whole algorithm consists in iterative applying these 3 steps (assignment of clusters, subspace determination and merging clusters) and terminates as soon as the current number of clusters equals the number k given as input parameter. This algorithm is very sensitive to the choice of k. If the input k was wrong the results worsen essentially. Besides, it is desirable that l is also quite near to the dimensionality of resulting clusters.

4C (Computing Correlation Connected Clusters) [23] represents an approach extending DBSCAN [46] with PCA. Input parameters for the approach are the dimensionality λ of computed correlation clusters, their jitter δ as well as extra parameters μ and ϵ determining cluster's minimum density. More precisely, λ is the density of hyperplanes formed by the neighborhood points of so called correlation core points, whereas the threshold δ determines the admissible deviation of these hyperplanes from the perfect ones. In this algorithm, each object is checked if it is a correlation core object or not. If the point is not a correlation core object, it is marked as noise and dropped. Otherwise, the approach tries to build a cluster for this correlation core object by adding to it all correlation-reachable objects. In such a way, the algorithms goes through all the correlation core points. The correlation clusters are sensitive to the choice of λ and correspondingly to δ.

HiCO(Hierarchical Correlation Ordering)[3] is a representative of correlation clustering. The method consists in searching for correlation in various features in high dimensional data and visualizing the built cluster hierarchy by means of correlation diagrams. Taken the correlation distance as distance function, one can recognize a high correlation between many attributes of two points if this special distance is small. Such two points are put therefore into a common cluster. Handling in this way, a hierarchy of clusters is constructed, where the clusters with short distances (small correlations) are nested into the clusters with longer distances (higher correlations).

ERiC[4](Exploring complex hierarchical Relationships among Correlation clusters) is a yet another technique searching for relationships between correlation clusters which however visualizes the resulting hierarchy in form of a graph-like structure. Unlike many hierarchical approaches, which use tree representation for the hierarchy, ERiC succeeds to find out also multiple inclusions. The method consists in generating all correlation clusters for all thinkable correlation dimensions, after which a hierarchical structure is built. There are 3 phases in the algorithm: partitioning w.r.t. correlation dimensionality, correlation clusters computing (within each partition) and, finally, aggregating the correlation clusters hierarchy.

10.3 Model-Based Hierarchical Clustering

10.3.1 AV-Link-Approach for Model-Based Hierarchical Clustering

To confirm our hypothesis that AD is an aging process, we want to compare the groups of patients and healthy controls of different age. Group comparison and relation analysis between groups is well studied by means of hierarchical clustering. The model, constructed in such a way, gives an idea how similar or distinct are the groups to each other. We want to see how close and in which hierarchy the different groups are to each other. We interested in how much the activity in the regions influenced by the disease distinguishes among healthy and ill persons of different age. We expect that the main differences in the activity of these regions gives the age, i.e. the groups of elder and ill persons are near to each other, whereas the young control group is far from them.

We adapt the classical AV-Link from book [90] for comparison the groups of patients and of healthy controls. What we are interested in, is to clear how close the group of aged healthy controls is to the group of the patients and to the group of young healthy controls. For that, we initialize the classical AV-Link with the given groups as initial clusters. It represents the distinction from the classical AV-Link, where the initial clusters consist of single objects from the dataset. Further, we calculate the distance between all pairs of clusters and merge the two closest clusters into a new one, constructing thus a dendrogram. This dendrogram differs from the classical one in the way that its leaves are the given groups and not the single objects.
The main innovation of our method consists in definition of distance between clusters. In this chapter we use our own model-based distance. To verify our hypothesis about the relationship between AD and aging process we use the classical euclidean distance. The persons in our research are represented by multidimensional time series. For classical euclidean approach we linearize the multidimensional time series in very long vectors. In the case when the time series length is different, we add zero padding to the affected vectors to align the length. Next, we calculate the distance between all the pairs of objects from the both clusters and take the average distance as the distance

between the clusters. Formally, the distance between C_1 and C_2 is defined as follows:

$$dist(C_1, C_2) := \frac{1}{|C_1||C_2|} \sum_{x \in C_1; y \in C_2} dist_{euclid}(x, y)$$

where the euclidean distance between the vectors x and y is defined as:

$$dist_{euclid}(x, y) := \sqrt{\sum_{i=1}^{length(x)} (x_i - y_i)^2}$$

Intercluster model-based distance we define in the following way.
We calculate the model-based distance between the clusters C_1 and C_2 as the average distance between the cluster and all the objects of the opposite cluster:

$$D(C_1, C_2) := \frac{1}{2}\left(\frac{1}{|C_1|} \sum_{X_* \in C_1} D_{C_2}(X_*) + \frac{1}{|C_2|} \sum_{X_* \in C_2} D_{C_1}(X_*) \right)$$

where $|C|$ is the number of objects in cluster. $D_C(X)$ is the distance from object X to cluster C. More precisely, it is the error of object X in the model of cluster C. In other words:

$$D_C(X) := \frac{1}{m} \sum_{1 \leq j \leq m} ||\varepsilon_j|| = \frac{1}{m} \sum_{1 \leq j \leq m} \left|\left| X_*^{(j)} - \sum_{1 \leq i \leq m; j \neq i} P_j^i X_*^i \right|\right|$$

Here m is the number of dimensions in our multidimensional time series. ε_j is the error in the model of the dimension j, X_*^j is the jth dimension of multidimensional time series X, i.e. $[X_*^1 \ldots X_*^m] := X$. P^i is the model for ith dimension. P_j^i is the jth coefficient of the linear combination of dimension i. P is calculated for the multidimensional time series cluster by means of the least squares method. The calculation of P and the optimization of these computations is described in detail in Section 9.3 and in the paper [155]. In this paragraph we are talking about model-based distance between clusters. In fact, $D(C_1, C_2)$ does nor represent a classical distance. E.g., for $D(C_1, C_2)$ does not hold the triangle inequality: $D(C_1, C_3) \leq D(C_1, C_2) + D(C_2, C_3)$, i.e. this "distance" is not transitive. This "distance" is besides also not reflexive: it is possible that $D(C, C) > 0$. In spite of these restrictions, the function proposed by us reflects well the relationships between multidimensional time series groups. In the following, we will call this relation distance, taking into account that this definition does not meet the classical requirements to the distance. To confirm our hypothesis we construct hierarchical clustering based on several representations of our data. We also carry out a permutation test, to show the relevance of our results.

10.3.2 Subspace Clustering over Length of Models

In the previous chapters we used heuristic methods (e.g. BIC) to select the dimensions we include into the linear combinations of model. These methods reached their limits when increasing the dimension number. In the approach proposed in this Chapter, searching for dimension subset ends when we reach the given length of the linear combination (what corresponds to model in this context). In this subsection we propose a hierarchical clustering where the linear combination length for each dimension (i.e. model) depends on the splitting depth (level) in the dendrogram. We split the cluster in two parts. For that, we use IKM with $k = 2$. Instead of BIC we choose a fixed number of summands in the linear combination. The same way as in IKM we use various random seed to find a good initialization.

To keep it simple: the summand number equals the level in the dendrogram as in Figure 10.1. We stop the hierarchy construction after a certain depth or minimal number of objects in new clusters is reached. The formal algorithm is presented in the Figure 10.2 in a form of pseudocode.
We proceed from the fact, that simple distinctions between clusters are seen directly and these distinctions are expressed by simple dependencies only on one dimension. The deeper the hierarchy we build is, the longer (more complex) the dependency becomes. This complex model is applied however for less number of objects, what makes the model more precise. We do not try to find a universal model for the whole cluster, but only refine the model for its subclusters.

10.3.3 Subspace Clustering over Number of Models

In comparison with the previous subsection, in this subsection we will search for models of arbitrary length, similar as in the case of IKM from Chapter 9. But at the different hierarchy depth we take various number of dimension models. That is, the dendrogram root representing our cluster is split by only one model (which is in the turn a linear combination). At the second level,

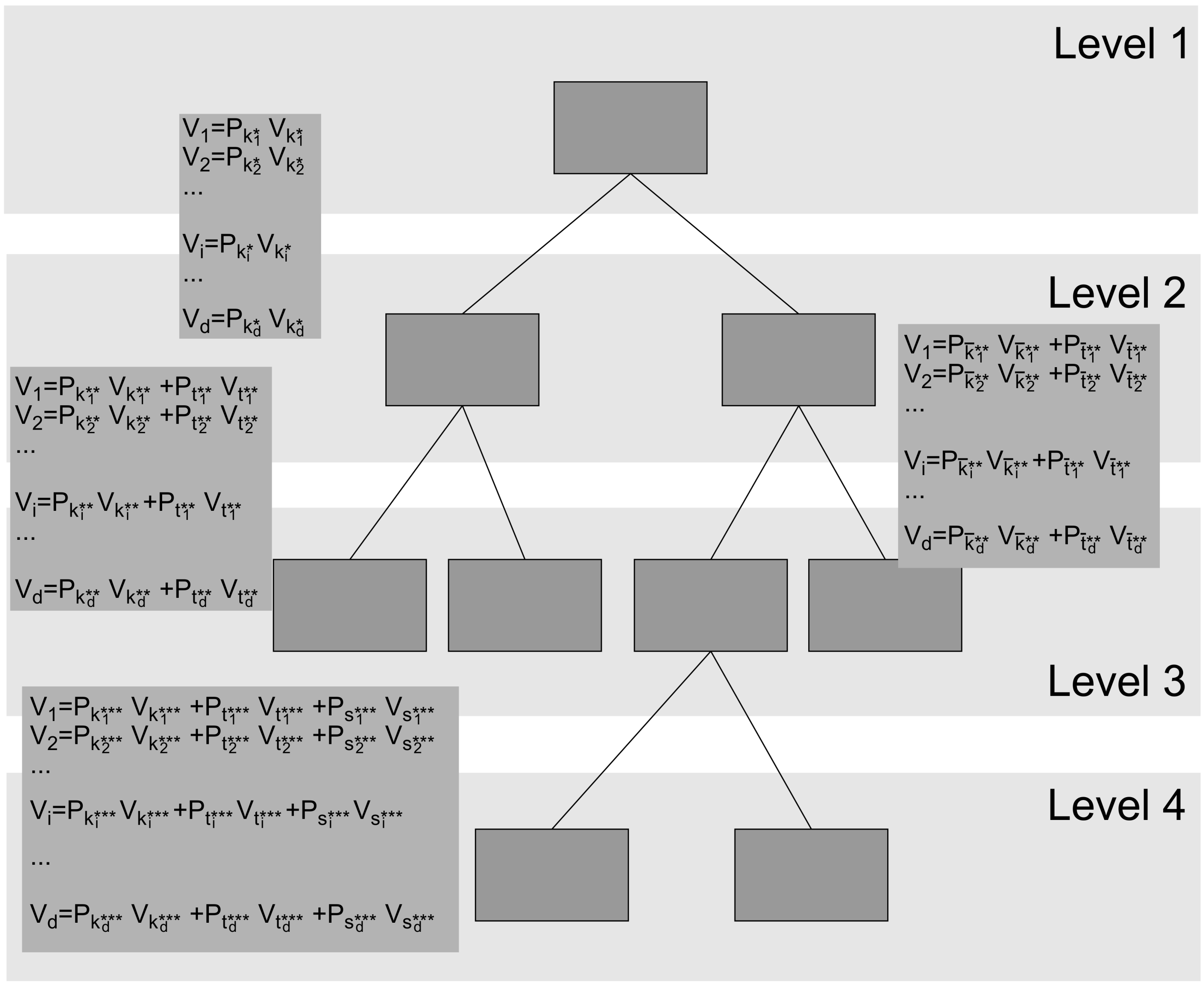

Level 1
$V_1=P_{k_1^*} V_{k_1^*}$
$V_2=P_{k_2^*} V_{k_2^*}$
...
$V_i=P_{k_i^*} V_{k_i^*}$
...
$V_d=P_{k_d^*} V_{k_d^*}$
Level 2
$V_1=P_{\bar{k}_1^{**}} V_{\bar{k}_1^{**}} +P_{\bar{t}_1^{**}} V_{\bar{t}_1^{**}}$
$V_2=P_{\bar{k}_2^{**}} V_{\bar{k}_2^{**}} +P_{\bar{t}_2^{**}} V_{\bar{t}_2^{**}}$
...
$V_i=P_{\bar{k}_i^{**}} V_{\bar{k}_i^{**}} +P_{\bar{t}_1^{**}} V_{\bar{t}_1^{**}}$
...
$V_d=P_{\bar{k}_d^{**}} V_{\bar{k}_d^{**}} +P_{\bar{t}_d^{**}} V_{\bar{t}_d^{**}}$
$V_1=P_{k_1^{**}} V_{k_1^{**}} +P_{t_1^{**}} V_{t_1^{**}}$
$V_2=P_{k_2^{**}} V_{k_2^{**}} +P_{t_2^{**}} V_{t_2^{**}}$
...
$V_i=P_{k_i^{**}} V_{k_i^{**}} +P_{t_1^{**}} V_{t_1^{**}}$
...
$V_d=P_{k_d^{**}} V_{k_d^{**}} +P_{t_d^{**}} V_{t_d^{**}}$
Level 3
$V_1=P_{k_1^{***}} V_{k_1^{***}} +P_{t_1^{***}} V_{t_1^{***}} +P_{s_1^{***}} V_{s_1^{***}}$
$V_2=P_{k_2^{***}} V_{k_2^{***}} +P_{t_2^{***}} V_{t_2^{***}} +P_{s_2^{***}} V_{s_2^{***}}$
...
$V_i=P_{k_i^{***}} V_{k_i^{***}} +P_{t_i^{***}} V_{t_i^{***}} +P_{s_i^{***}} V_{s_i^{***}}$
...
$V_d=P_{k_d^{***}} V_{k_d^{***}} +P_{t_d^{***}} V_{t_d^{***}} +P_{s_d^{***}} V_{s_d^{***}}$
Level 4

Figure 10.1: Meta-Model for hierarchical clustering over length of models.

```
[ClusteringOverLengt]
Input: data set DS
Output: Clustering C
  Clustering $bestClustering$;
  //The maximum depth of the dendrogram.
  maxDepth = config.getMaxDepth;
  //The maximum number of the iterations
  // to find a optimal clustering.
  maxIter = config.getMaxIterations;
  $currCl = 1$;
  Clustering $Cl = DS$;
  //while desired number of splits not reached
  while $k < maxDepth$ do
    $dimNum = level\ of\ leaf$
    //It will be tried $maxInit$ random initialization.
    for $i = 1 \rightarrow maxInit$ do
      //random split of cluster
      $Cl.add = randomSplit(Cl.get(currCl))$
      iter = 0;
      while (not converged) and (iter++ < maxIter) do
        for each object $O \in Cl.get(currCl)$ do
          assign $O$ to the child cluster where error is minimal
          for each $leaf\ L \in C$ do
            //recalculate model
            $M_L = findModels(L, dimNum)$
          end for
        end for
        if improvement of object function then
          //memorize best Clustering
          $bestClustering = Cl$;
        end if
      end while
    end for
    $currCl = currCl + 1$;
  end while
  return $C := bestClustering$
```

Figure 10.2: Subspace Clustering over length of models

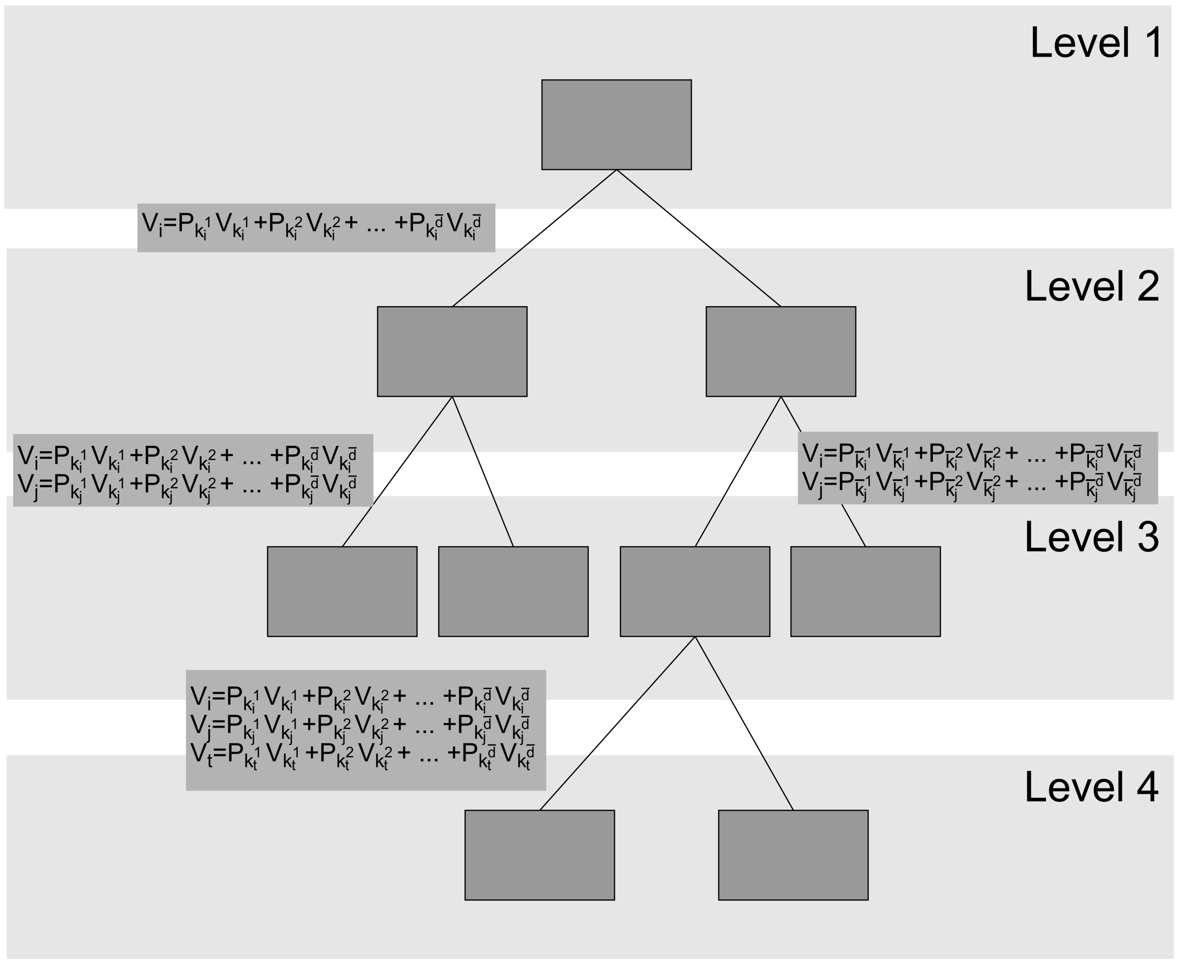

Level 1
$V_i=P_{k_i^1}V_{k_i^1}+P_{k_i^2}V_{k_i^2}+\dots+P_{k_i^{\bar d}}V_{k_i^{\bar d}}$
Level 2
$V_i=P_{k_i^1}V_{k_i^1}+P_{k_i^2}V_{k_i^2}+\dots+P_{k_i^{\bar d}}V_{k_i^{\bar d}}$
$V_j=P_{k_j^1}V_{k_j^1}+P_{k_j^2}V_{k_j^2}+\dots+P_{k_j^{\bar d}}V_{k_j^{\bar d}}$
$V_i=P_{\bar k_i^1}V_{\bar k_i^1}+P_{\bar k_i^2}V_{\bar k_i^2}+\dots+P_{\bar k_i^{\bar d}}V_{\bar k_i^{\bar d}}$
$V_j=P_{\bar k_j^1}V_{\bar k_j^1}+P_{\bar k_j^2}V_{\bar k_j^2}+\dots+P_{\bar k_j^{\bar d}}V_{\bar k_j^{\bar d}}$
Level 3
$V_i=P_{k_i^1}V_{k_i^1}+P_{k_i^2}V_{k_i^2}+\dots+P_{k_i^{\bar d}}V_{k_i^{\bar d}}$
$V_j=P_{k_j^1}V_{k_j^1}+P_{k_j^2}V_{k_j^2}+\dots+P_{k_j^{\bar d}}V_{k_j^{\bar d}}$
$V_t=P_{k_t^1}V_{k_t^1}+P_{k_t^2}V_{k_t^2}+\dots+P_{k_t^{\bar d}}V_{k_t^{\bar d}}$
Level 4

Figure 10.3: Meta-Model for hierarchical clustering over number of models.

```
[RankingModel]
Input: List of all Models mods
Output: List of best Models $MB$
  $errors = new\ array[mods]$
  for each $Object\ o \in Cluster$ do
    for each $Model\ m \in mods$ do
      //summerize errors of $o$ and $m$
      $errors[m] = errors[m] + error_o$
    end for
  end for
  sort errors in ascending order
  for $i = 0 \to dimNum$ do
    save corresponding model of errors[i] in MB
  end for
```

Figure 10.4: Ranking Best Models

we choose two dimensions, using by that two models and so on. In other words, the number of models in subclusters equals the depth, at which cluster splitting occurs. The clustering model structure is shown in the Figure 10.3. We select the dimension for splitting according to the rule of the most accurate model. That is, the less the error in the model is, the more precise this model is and the better it should describe (and split) its subclusters. We use a helper method *RankingModel* (see Figure 10.4) for choosing the most accurate models. We calculate the average error thereby for all models.
Subspace clustering over number of models algorithm is similar to the previous one, but it gets complicated by the search for interesting models.

The problem is that the linear combination consists of several dimensions. Because of imperfection of BIC, we cannot be sure, that in the linear combination together with all dependent dimensions we did not chose erroneously also independent dimensions. When choosing the most precise models for several dimensions we possibly find only one dependency between several dimensions and choose these dimensions as the most precise. Our aim is however to find a new dependency from another dimensions. To solve this problem we need to find all dimensions taking part in a linear combination.

To do that, we calculate the model P_i for each dimension V_i. Our linear combination:

$$V_i = P_{k_i^1} V_{k_i^1} + P_{k_i^2} V_{k_i^2} + \cdots + P_{k_i^d} V_{k_i^d}$$

can be rewritten as:

$$P_{k_i^1} V_{k_i^1} + P_{k_i^2} V_{k_i^2} + \cdots + (-1)V_i \cdots + P_{k_i^d} V_{k_i^d} = 0$$

Next, we normalize the coefficients of resulting equalities (i.e. linear combinations). Now, we can calculate euclidean distance between the coefficients of two equalities. If these equalities describe the same dependency, then they should be very similar and the euclidean distance between them should be very small. For the models, more precisely for their equalities from different dependencies, the coefficients should differ essentially, what the euclidean distance in essence reflects. We use a threshold T for filtering already used dependencies. If a similar dependency (the one having euclidean distance between equalities less than T) was already used, then we exclude this dimension from the list of dimensions we use in the future for splitting in this dendrogram branch. For selecting the models we use two heuristics described in the following subsections.

Distance between Child Clusters

We search for subclusters and models, that split these clusters in an optimal way. The models found in each subcluster after each iteration are compared to each other and the similar models are dropped. In such a way, we choose the models with the least error, i.e. the most accurate ones. At the same time, our models should be good in splitting cluster into subclusters. The method is presented in the Figure 10.5.

Distance between Child Cluster and Parent Cluster

Here we compare the models of child cluster AM with the models of its parent cluster P. We filter all dependencies, that have been already used in parent cluster. The most distinct models we will use to split parent cluster into child subclusters. Here we are going to find new not yet used dependencies, which split thereby the cluster in the best way. The algorithm for that is depicted in the Figure 10.6

```
[ClusteringOverNumofModV1]
Input: data set DS
Output: Clustering bestClustering
  $Clusterc = DS$;
  Clustering $C = c$
  $currCl = 1$;
  for $seed = 1 \cdots MaxSeedNumber$ do
    $C.get(currCl).randomSplit(seed)$
    while not converged or iter < maxIter do
      for each object $O \in C.get(currCl)$ do
        assign O to the cluster where error is minimal
        for each $leaf\ L \in C$ do
          $AM = findModels(L)$
          filteredModels = filterMostSimilarModels(AM)
          $M_L = getBestModels(filteredModels, dimNum)$
        end for
      end for
      if improvement of object function then
        $bestClustering = C$;
      end if
    end while
  end for
```

Figure 10.5: Subspace Clustering over number of Dimensions. Comparing models of child Clusters

```
[ClusteringOverNumofModV2]
Input: data set DS
Output: Clustering bestClustering
  $Clusterc = DS$;
  Clustering $C = c$
  $currCl = 1$;
  for $seed = 1 \cdots MaxSeedNumber$ do
    $C.get(currCl).randomSplit(seed)$
    while not converged or iter < maxIter do
      for each object $O \in C.get(currCl)$ do
        assign O to the cluster where error is minimal
        for each leaf $L \in C$ do
          $AM = findModels(L)$
          $M_L$ = getMostDissimilarModels(P, AM, dimNum)
        end for
      end for
      if improvement of object function then
        $bestClustering = C$;
      end if
    end while
  end for
```

Figure 10.6: Subspace Clustering over number of Dimensions. Comparing models between child and parent

10.4 Evaluation

10.4.1 Analysis of Medical Data with Model-Based Hierarchical Clustering

We use different representation of our subjects. For the chosen regions we take the mean or IC which corresponds to the given region. Also we apply our algorithm to different subsets of regions. Whereas each region is represented by time series, a subset of regions of a subject forms a multivariate time series. To validate the results we perform multiple (10 000 times) permutation test.

Using automated network selection based on network maps described above [9], ICA of rs-fcMRI data through the subjects of model order 75 detected totally 22 intrinsic networks of interest. Among these networks there are four DMN, four attentional, three frontal, four visual, one auditory, five somatosensory and one basal ganglia sub-networks, accordant with previous studies [9] (Figure 10.7, Table 13.2; Table 13.2 for 55 peak voxels of significant clusters; $p < 0.05$, FWE cluster level corrected).

Similarity of Inter-Subject-Synchronicity Estimated by Average Linkage Clustering

We applied Average Linkage clustering (with Euclidean distance) to different ensembles of iRA (Figure 10.8) and respectively iNA (Figure 10.9) to estimate the similarity of inter-subject-synchronicity (ISS) [191] across the groups. In the case of iNA, older healthy controls were consistently more similar to patients than to young healthy persons, indicating increased similarity of ISS between patients and older persons. Resulting output was independent from chosen network ensembles (Figure 10.9 A-C). Average Linkage clustering of iRA showed a similar situation, i.e. older healthy controls were more similar to patients than to young healthy persons (Figure 10.9 A-C). At the same time for primary networks' iRA, older healthy participants were clustered closer to younger healthy controls than to patients (Figure 10.9 C).

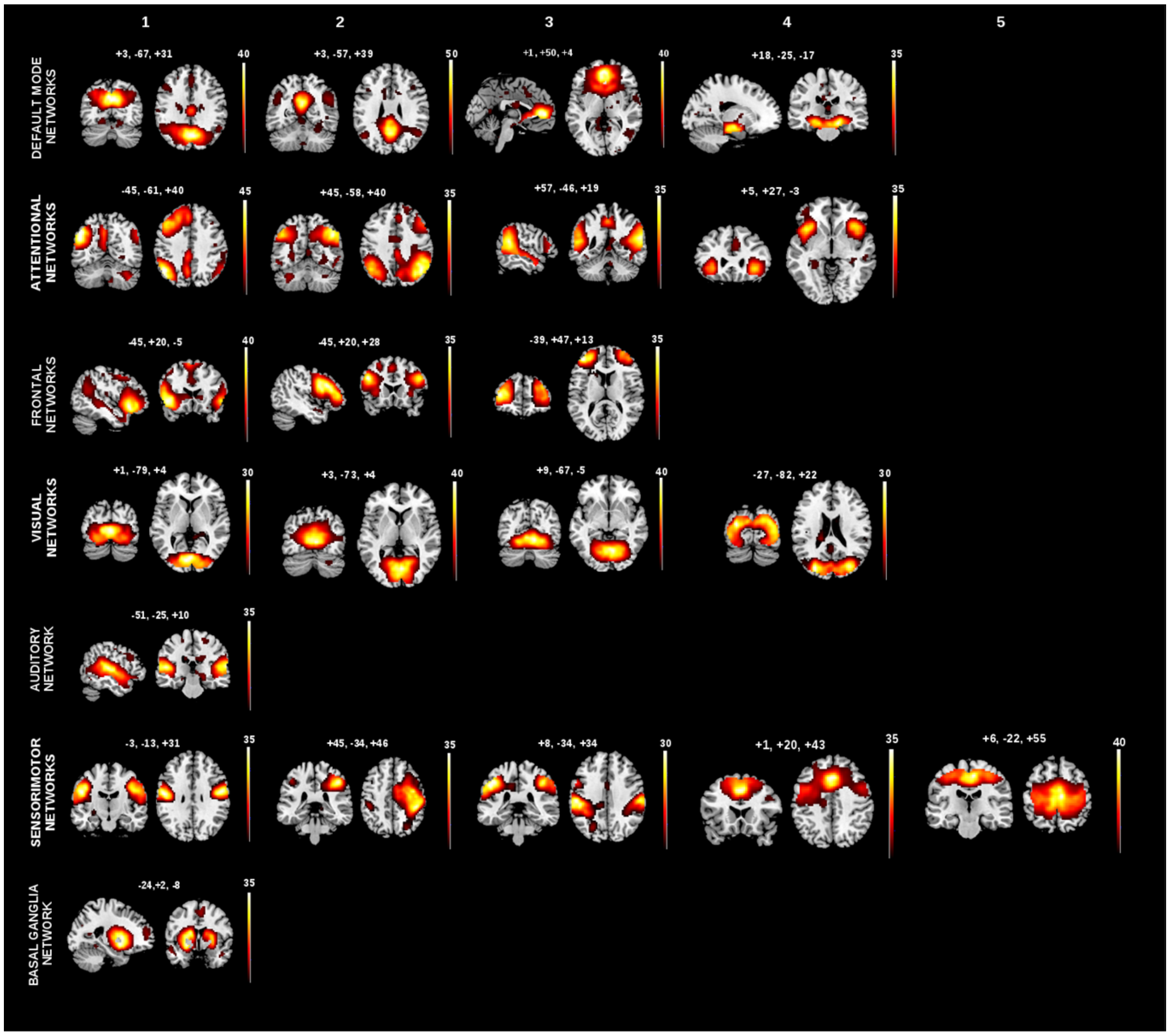


Figure 10.7: Spatial maps of intrinsic networks. Here are presented the patterns derived from spatial ICA of rs-fMRI data based on the following groups: AD patients, mild cognitive impairment as well as older and younger healthy controls. Using spatial regression with canonical network templates, 22 components were totally identified as intrinsic networks, including four default mode networks, four attentional networks, three frontal networks, four visual networks, one auditory network, five somatosensory networks and one basal ganglia network. The results of one-sample t-test on the individual back-reconstructed subject component patterns across patients and controls ($p < 0.05$, cluster level family wise error corrected) is depicted above in the form of color maps, which are superimposed on a single-subject $T1$ image. $t-$values are represented by red to yellow scales.

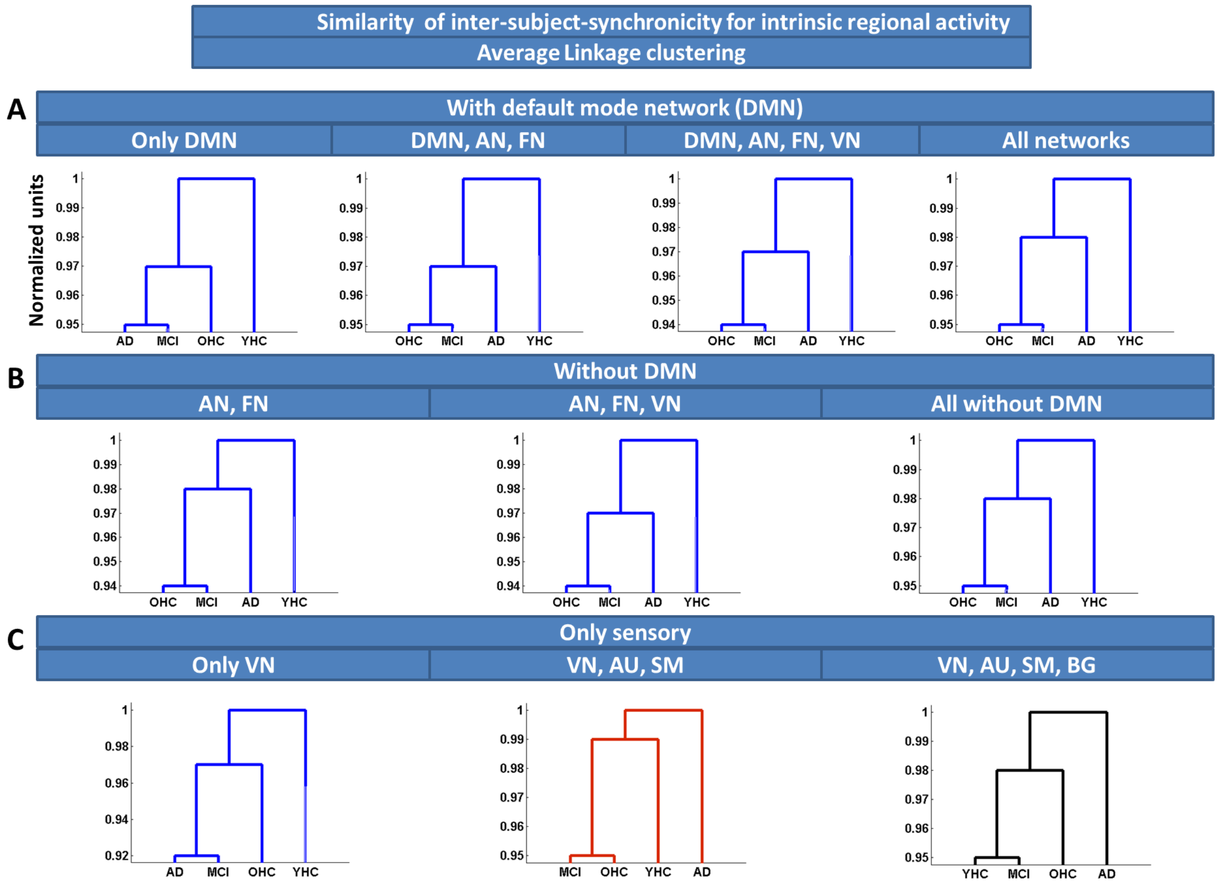


Figure 10.8: Inter-subject-synchronicity similarity for intrinsic regional activity, estimated by means of Average Linkage clustering based on Euclidean distance. Different regional ensembles were used during clustering: A) ensembles that include regions of the DMN; B) ensembles that exclude regions of the DMN; C) ensembles of primary and subcortical regions. Independent of the ensemble, older healthy controls were grouped according to clustering closer to patients than to younger healthy controls (blue dendrograms). Scale (in arbitrary units) reflects normalized distances among clusters. AD= Alzheimer's disease, AN= attentional networks, AU= auditory network, BG = basal ganglia networks, DMN= default mode networks, FN= frontal networks, MCI = mild cognitive impairment, OHC= older healthy controls, SM = somatomotor networks, VN= visual networks, YHC= younger healthy controls.

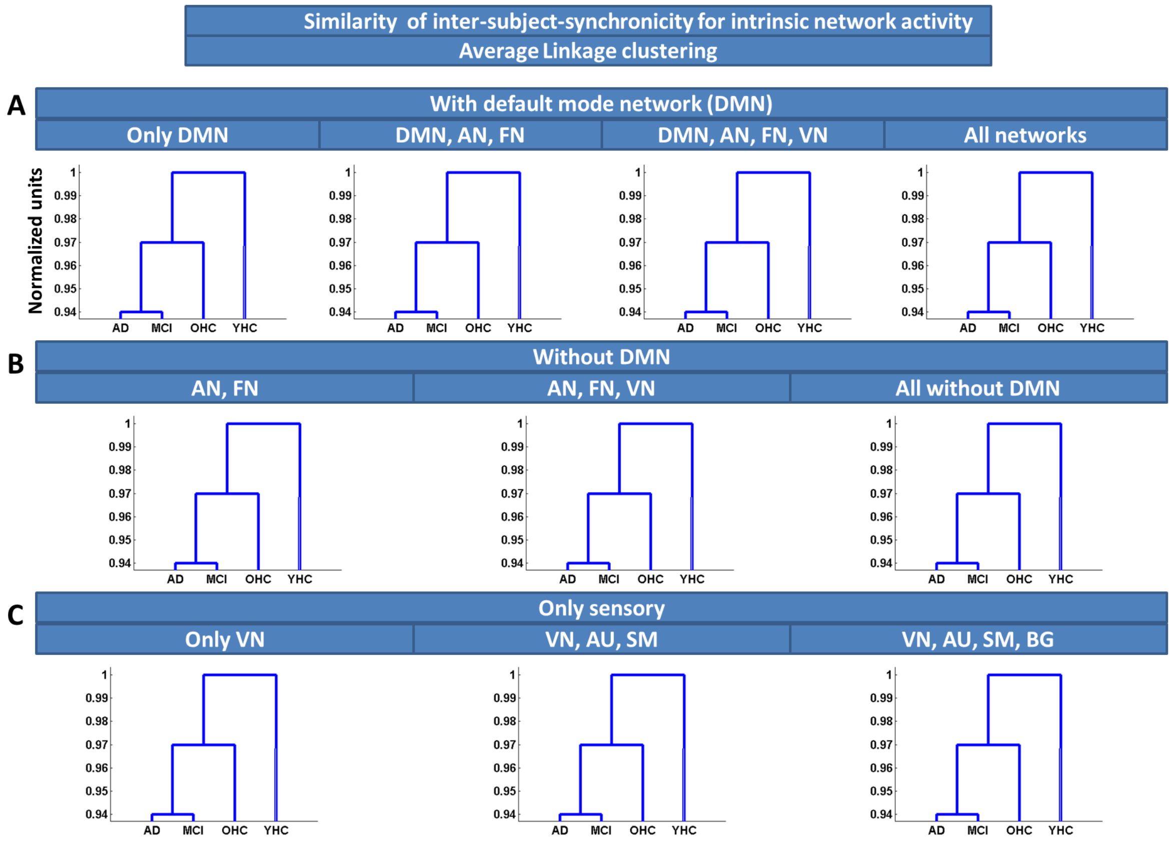


Figure 10.9: Inter-subject-synchronicity similarity for intrinsic network activity, estimated by means of Average Linkage clustering based on Euclidean distance. Different network ensembles were used during clustering: A) ensembles that include the DMN; B) ensembles that exclude the DMN; C) ensembles of primary and subcortical networks. Independent of the ensemble, older healthy controls were grouped according to clustering closer to patients than to younger healthy controls (blue dendrograms). Scale (in arbitrary units) reflects normalized distances among clusters. AD= Alzheimer's disease, AN= attentional networks, AU= auditory network, BG = basal ganglia networks, DMN= default mode networks, FN= frontal networks, MCI = mild cognitive impairment, OHC= older healthy controls, SM = somatomotor networks, VN= visual networks, YHC= younger healthy controls.

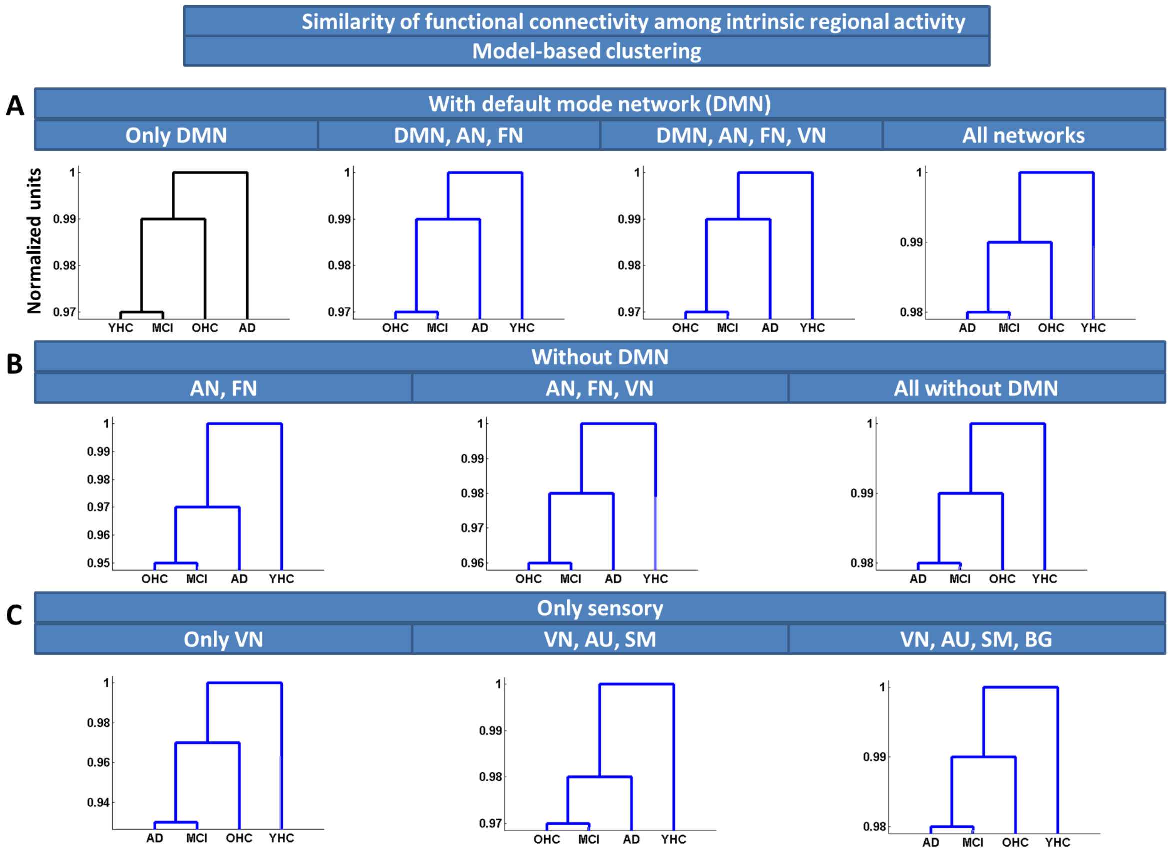


Figure 10.10: Functional connectivity similarity for intrinsic regional activity, estimated by model-based clustering based on linear combinations of time courses. Different regional ensembles were used during clustering: A) ensembles that include regions of the DMN; B) ensembles that exclude regions of the DMN; C) ensembles of primary and subcortical regions. Independent of the ensemble, older healthy controls were grouped according to clustering closer to patients than to younger healthy controls (blue dendrograms). Scale (in arbitrary units) reflects normalized distances among clusters. AD= Alzheimer's disease, AN= attentional networks, AU= auditory network, BG = basal ganglia networks, DMN= default mode networks, FN= frontal networks, MCI = mild cognitive impairment, OHC= older healthy controls, SM = somatomotor networks, VN= visual networks, YHC= younger healthy controls.

Similarity of Functional Connectivity Estimated by Model-Based Clustering

We applied model-based clustering of iRA and respectively iNA, to investigate the similarity of intrinsic FC via linear combinations of activity time courses. In the case of iNA, older healthy controls were rather similar to patients than to young healthy persons, suggesting increased similarity of FC between patients and older persons. Resulting Clustering was independent of the chosen network ensembles (Figure 10.11 A-C). Model-based clustering of iRA detected a similar situation, where older healthy controls are more similar to patients than to young healthy participants (Figure 10.10 A-C) with only one exception for the DMN see Figure 10.10 A).
To estimate the similarity of intrinsic brain activity along aging and AD, we applied rs-fMRI and hierarchical clustering for healthy younger and older persons as well as patients with MCI and AD dementia. Independently of measures or regional sources of intrinsic activity, intrinsic activity of 70-years persons showed more similarities with that of patients with AD than of younger controls, potentially because of a significant proportion of preclinical AD cases in the group of healthy older persons.

10.4.2 Hierarchical Subspace Clustering

Here we make a proof of concept of proposed algorithms for Hierarchical Subspace Clustering. For that, we generate synthetic objects, whose time series fit perfectly to the models of methods we proposed. Besides, we show noise-robustness of these methods. Finally we demonstrate our algorithm for real world data. To estimate the quality of resulting clustering we apply to the dendrogram leaves well-known methods as in AMI[195], NMI[195] and Cluster Purity[199].
For Hierarchical Subspace Clustering over length of model, our objects have the length of 3,333 time points and consist of 12 dimensions. We generate 8 clusters which correspond to dendrogram leaves. Each cluster includes 100 objects. The clusters are built in the way that they specify 4-level dendrogram. For synthetic data without noise, the method gives very good results. The cluster purity is very close to 100%. AMI and NMI at the same time are near to 1. With increasing of number of noise objects, the quality decreases and at 10% noise it falls down to 49% for cluster purity and to 0.56 correspondingly 0.55 for NMI and AMI. In the Figure 10.13, we can see very

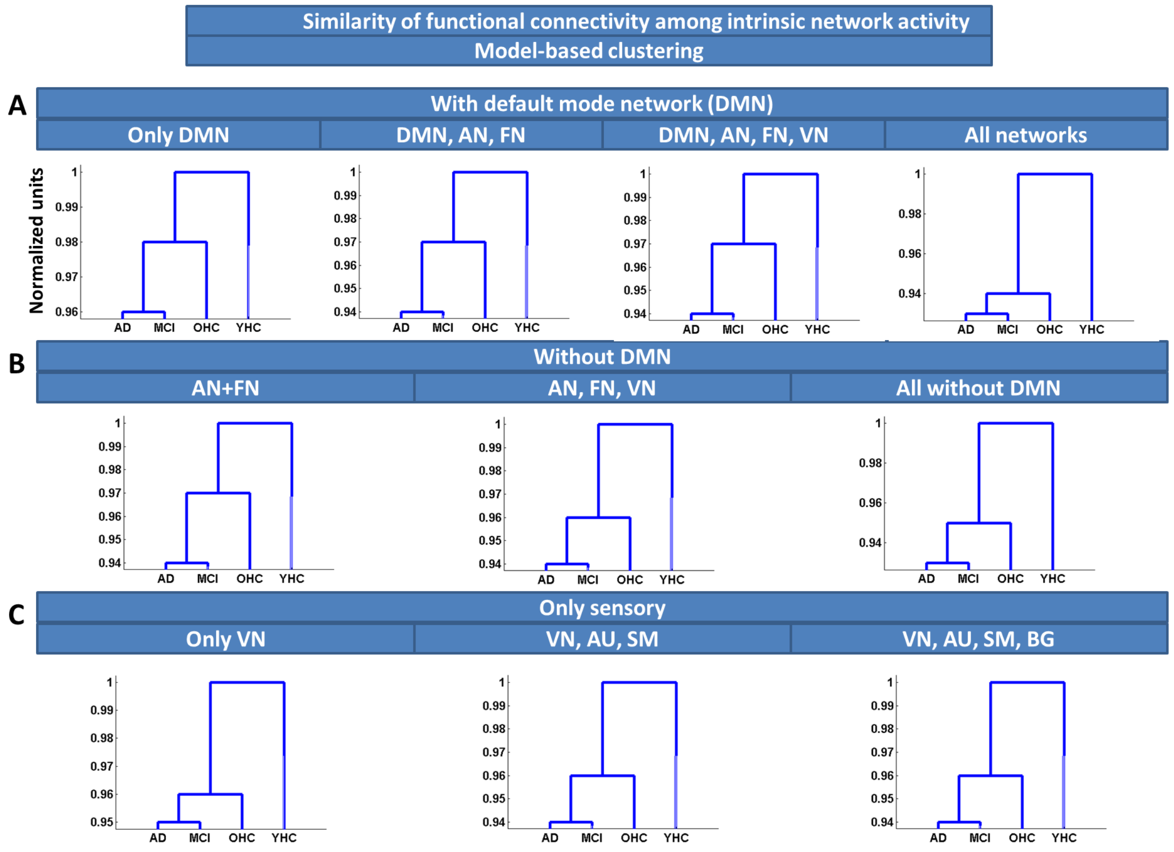


Figure 10.11: Functional connectivity similarity for intrinsic network activity, estimated by model-based clustering based on linear combinations of time courses. Different network ensembles were used during clustering:: A) ensembles that include the DMN; B) ensembles that exclude the DMN; C) ensembles of primary and subcortical networks. Independent of the ensemble, older healthy controls were grouped according to clustering closer to patients than to younger healthy controls (blue dendrograms). Scale (in arbitrary units) reflects normalized distances among clusters. AD= Alzheimer's disease, AN= attentional networks, AU= auditory network, BG = basal ganglia networks, DMN= default mode networks, FN= frontal networks, MCI = mild cognitive impairment, OHC= older healthy controls, SM = somatomotor networks, VN= visual networks, YHC= younger healthy controls

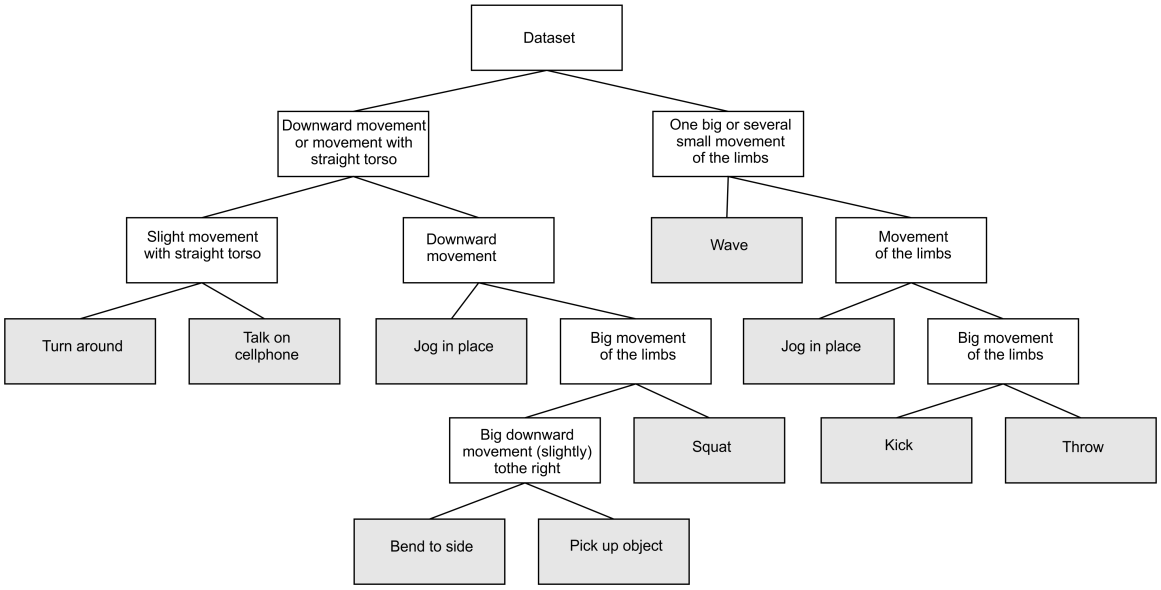

Dataset
Downward movement or movement with straight torso
One big or several small movement of the limbs
Slight movement with straight torso
Downward movement
Wave
Movement of the limbs
Turn around
Talk on cellphone
Jog in place
Big movement of the limbs
Jog in place
Big movement of the limbs
Big downward movement (slightly) tothe right
Squat
Kick
Throw
Bend to side
Pick up object

Figure 10.12: The result of Subspace Clustering w.r.t the model lengths. The original clusters from DS6 are depicted in grey color.

good results of our algorithm even for essential noise level.

For Hierarchical Subspace Clustering over number of models, we generate data of the same dimension as in the previous paragraph, but with models that perfectly suit this method model. The result quality after three splits is less than 30% cluster purity. This is noticeably worse than the algorithm from the previous paragraph, but noticeably better than random distribution into 8 clusters. We think, that the reason for such big inaccuracy is the choice problem of reasonable subset of dimensions at each step. This is the problem of exponential complexity and the both of our heuristics are not always able to choose the needed dependency. We did not test this approach neither on data with noise nor on real-world data.

Real data

We took DS6 and DS7 from Section 9.7. Dataset are described in the Table 9.1. DS6 can be easily interpreted, since each object represents a short video record of a certain motion. On DS6 we reached cluster purity 79%, NMI 0.78 and AMI 0.72. The resulting hierarchical clustering is presented in the Figure 10.12. The inner clusters of dendrogram have very reasonable label. Some leaves (one on the level 3 and one on the level 4) have cluster purity 100%. Two further leaves at the level 4 have a slightly worse but also very good purity 90%, which corresponds to the situation when solely one object in cluster is assigned to wrong cluster. The last leaf at level 4 has a purity of 70%. The leaves at the levels 5 and one leaf at level 6 have the purity of 80% or 70%. One cluster at level 6 only has a purity of 50% which means that only half of the objects are correctly classified.
For DS7 we got NMI/AMI 0.69/068. The mean cluster purity of the clustering is 76%. The resulting dendrogram is given in the Figure 10.14. In comparison with the DS6, the clusters with the highest purities are located in the center of the clustering. The purities are between 93.24%, representing the highest value of this clustering, and 73.73%. The cluster at level 3 only has a purity of 65%. Finally, the clustering having the lowest purity is at level 5 and has a purity of only 45.76%.

10.5 Conclusion

In this Chapter we showed, that model-based clustering can be also successfully applied for search of hierarchical clustering in different fields of science

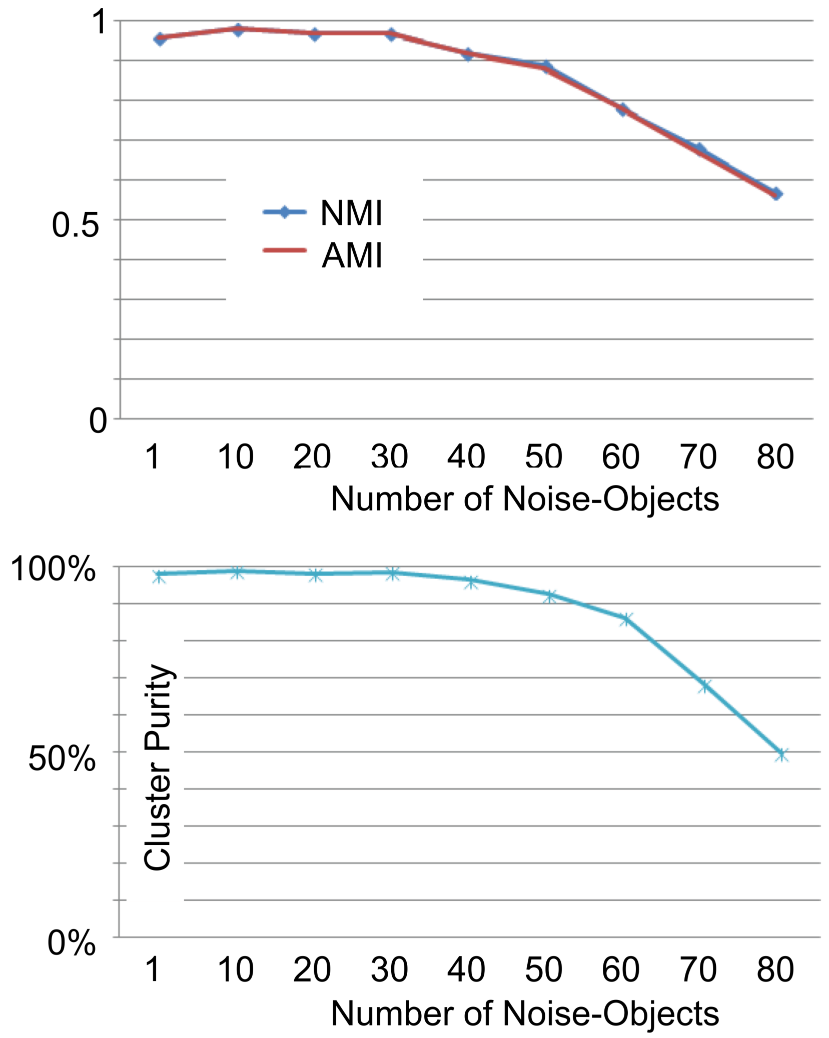


Figure 10.13: The result of Hierarchical Subspace Clustering over the length of models. Robustness to noise-objects in data set.

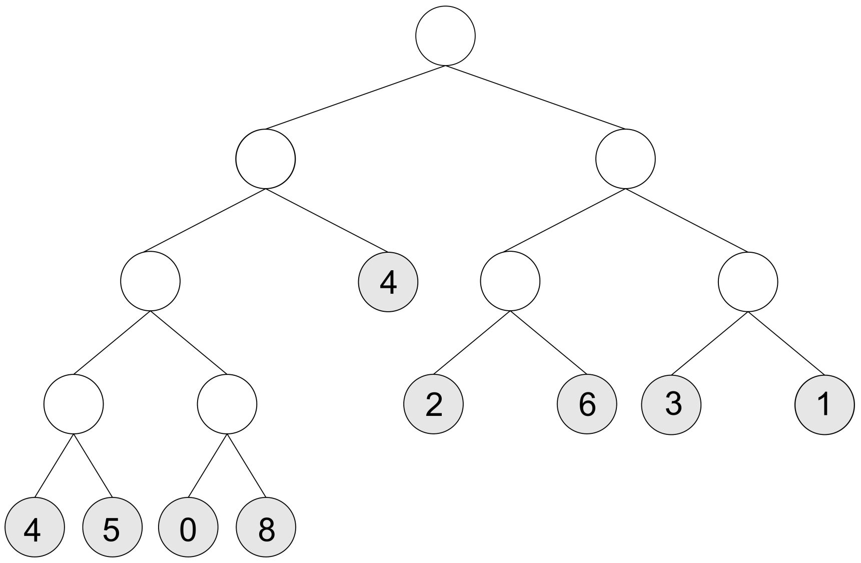


Figure 10.14: Result of Hierarchical Subspace Clustering over length of models. Grey color corresponds here to the original clusters from DS7.

and industry. We showed also that ideas of agglomerative and subspace hierarchical clustering is extensible to the mathematical models proposed by us. Because of opportunity to interpret the models we get a powerful tool for medical data analysis. By means of this method we found the dependency in motion video sequences and confirmed the hypothesis about a Alzheimer reason. Intrinsic brain activity of healthy elder persons resembles more that of patients with Alzheimer's disease than that of healthy younger persons. This finding suggests a significant proportion of preclinical AD cases in the group of cognitively normal older people. The link of aging and AD with intrinsic brain activity supports the view that lifespan intrinsic activity may contribute critically to the pathogenesis of AD. Algorithms showed not only interesting results on real-world data, but also a good noise-robustness.
We also saw some limits of our approaches. Agglomerative hierarchical clustering requires predefined clusters for further dendrogram constructing. Also the models, based on big number of subclusters will have a worse accuracy and essentially worse describe large groups of objects. A decision is here needed, how to improve the quality of models, describing a big number of subclusters.

We should note, that Subspace clustering over number of models did not show the expected result. In this algorithm we need to improve dimensions selection for further splitting. During model based classifying in Chapter 8 we also selected only interesting for us dimensions and reached excellent results. Subspace clustering over length of models demonstrated very interesting results. However, we want to find a better heuristics, which at noise level over 10% would give us a better result. An improvement target is also the distribution of noise objects within of inner clusters of hierarchy, if the objects do not fit subcluster models. In the future we want to find automatic methods for determination of reasonable depth of hierarchy and automatic solution concerning the appropriateness of splitting for a given cluster.
We plan to refine our algorithms for medical researches, since our methods take into account the requirements to inaccuracy in medical data and huge data volumes generated during medical studies.

Chapter 11

Decoding an Individual's Sensitivity to Pain from the Multivariate Analysis of EEG Data

The perception of pain is described by its tremendous intra- and interindividual variability. The same painful event can be perceived by different individuals largely differently. Our aim here is to predict the individual pain sensitivity from brain activity. We repeatedly applied identical painful stimuli to healthy human subjects and recorded brain activity by using electroencephalography (EEG). We applied a multivariate pattern analysis to the time-frequency transformed single-trial EEG responses. Our results show that a classifier trained on a group of healthy individuals can predict another individual's pain sensitivity with an accuracy of 83%. Classification accuracy depended on pain-evoked responses at about 8 Hz and pain-induced gamma oscillations at about 80 Hz. These results reveal that the temporal-spectral pattern of pain-related neuronal responses provides valuable information about the perception of pain. Beyond, our approach may be very helpful for establishing an objective neuronal marker of pain sensitivity which can potentially be recorded from a single EEG electrode.
Main parts of the material presented in this chapter have been published in Paper [169]. In this paper, Andrew Zherdin has proposed and implemented the workflow. Claudia Plant has optimized and improved the proposed pro-

cesses. The other coauthors have prepared data and written the medical part of the paper.

11.1 Introduction

Pain represents a complex sensory experience, which is perceived and described by a person not only in terms of the sensory information but also on the basis of the individual sensitivity to pain. In such a way, a certain event can be interpreted by one person as moderately or even slightly painful whereas for another individual the same event can cause a very strong pain experience [42, 109, 139]. Deviations in the sensitivity to pain can affect future pain experiences, responses to analgesic treatment [45, 139] as well as lead to development of chronic pain syndromes. Thus, knowing an objective neuronal marker of pain sensitivity could help in prevention, diagnosis and treatment of painful conditions [27, 187].

In the brain, the complex perception of pain is subserved by an extended network of brain areas [13, 188]. Recent neurophysiological studies disclosed different partially overlapping pain-related neuronal responses within this network. These pain-related neuronal responses include evoked responses at theta frequencies (3 - 8 Hz) [55, 120] and induced responses at gamma ($\sim 80Hz$) [63, 73] and alpha ($\sim 10Hz$) [136, 157] frequencies. A few studies related differences in pain sensitivity to brain activity. A functional magnetic resonance imaging (fMRI) study indicated that pain-related BOLD responses in somatosensory, anterior cingulate, and prefrontal regions reflect the subject's sensitivity to pain [42]. Neurophysiological studies showed that the individual pain sensitivity correlates with amplitudes of evoked responses at theta frequencies [87, 168]. It should, thus, in principle be possible to infer an individual's sensitivity to pain from pain-related brain responses.
Considering the multitude of pain-related neuronal responses, multivariate approaches which analyze complex patterns of information appear particularly promising for decoding the sensitivity to pain from brain activity. Such multivariate 'brain reading' approaches have recently been used to identify patterns of brain activity that differentiate between mental states [77, 121, 141]. Researchers have successfully inferred visual percepts, speech content or even hidden intentions from functional magnetic resonance imaging (fMRI) data (e.g. [51, 75, 76, 98]). A recent study applied multivariate

pattern analysis (MVPA) to pain and showed that it is possible to distinguish different levels of painful and non-painful stimulation based on fMRI data [128]. However, that study pursued an intraindividual approach, i.e. a classifier was trained on and applied to data from the same subject. In contrast, most practical applications require cross-subject approaches, i.e. a classifier is trained on a group of subjects and then used to predict another individual's mental state. Moreover, it would be particularly desirable to not only distinguish different stimuli but to infer how different individuals perceive objectively identical stimuli from brain activity.
Here, we therefore applied identical painful stimuli to a group of healthy human subjects. In a cross-subject 'brain reading' approach we aimed to decode an individual's pain sensitivity from the temporal-spectral pattern of neuronal responses as assessed by time-frequency transformed electroencephalographic (EEG) data. To this end, a MVPA technique was trained on the single trial EEG data of healthy human subjects to investigate whether it is possible to predict another individual's pain sensitivity from brain activity.

11.2 Methods

11.2.1 Paradigm

60 painful cutaneous laser stimuli of identical intensity were delivered to the dorsum of the right hand. The laser device was a Nd:YAP laser (Electronical Engineering, Florence, Italy) with a wavelength of 1340 nm, a pulse duration of 3 ms and a spot diameter of 6 mm. Stimulus intensity was kept constant at 2750 mJ, which evoked slightly to moderately painful pinprick-like sensations. Stimulation site was slightly changed after each stimulus. Interstimulus intervals (ISI) were randomly varied between 8 and 12 seconds. The subjects passively perceived the stimuli with closed eyes. Three seconds after stimulus application, the subjects were prompted by an auditory cue to verbally rate the pain intensity on a numerical rating scale between 0 (no pain) and 10 (maximum tolerable pain). Pain ratings were used to assign 'true' labels of pain sensitivity to each individual. To this end, trials were split around the mean of the 11-point numerical rating scale. Subjects who rated > 5 (high pain) more often than ≤ 5 (low pain) were labeled pain sensitive and subjects who rated ≤ 5 more often than > 5 were labeled pain insensitive.

11.2.2 EEG Recordings and Analysis

EEG data were recorded using an electrode cap (EASYCAP, Herrsching, Germany). The electrode montage included 64 electrodes consisting of all 10-20 system electrodes and the additional electrodes Fpz, FCz, CPz, POz, Oz, Iz, AF3/4, F5/6, FC1/2/3/4/5/6, FT7/8/9/10, C1/2/5/6, CP1/2/3/4/5, TP7/8/9/10, P5/6, PO1/2/9/10, plus two electrodes below the outer canthus of each eye. The EEG was referenced to the FCz electrode, grounded at AFz, sampled at 1 kHz ($0.1\mu V$ resolution) and highpass-filtered at 0.1 Hz. The impedance was kept below $20k\Omega$.
The raw EEG data were preprocessed in Vision Analyzer software (Brain Products, Munich, Germany) including downsampling to 512 Hz for the purpose of data reduction, correcting for horizontal and vertical eye movements using an independent component analysis [92], and transforming to the average reference [111]. Trials with artifacts exceeding $\pm 100\mu V$ in any channel were automatically rejected. The remaining trials were epoched from -1100 ms to 1500 ms. Evoked potentials (EPs) were computed by averaging the epochs. Single trial data as well as EPs were exported for subsequent processing to Matlab (The Mathworks, Natick, USA).
Time-frequency analyses were performed in Matlab using custom programming on the basis of standard mathematical and signal analysis functions. To compute time-frequency representations (TFR) we applied a single trial Hamming tapered, moving window short time Fast Fourier Transformation (FFT). The window had a length of 100 data points, was padded with zeros up to 512 data points and was shifted for 1 data point. Hence, frequency resolution was 1 Hz and temporal resolution was 1/512 sec. For each trial and electrode, the baseline corrected TFRs were computed and transformed into percent signal change with respect to the respective baseline from -1000 ms to 0 ms.

11.2.3 Multivariate Pattern Analysis (MVPA)

To predict an individual's sensitivity to pain from brain activity, we applied MVPA to time-frequency transformed EEG responses to pain. To make use of the full single trial information we applied a two-step approach. First, we used MVPA to predict single trial labels (≤ 5, low pain; > 5, high pain) from the single trial EEG responses. Second, we applied a voting-based approach

to predict an individual's sensitivity to pain (pain sensitive or pain insensitive) from the proportion of low pain and high pain trials. If more high pain trials than low pain trials were predicted the individual was classified as pain sensitive and vice versa. The MVPA was performed separately for each electrode. Importantly, we pursued a cross-subject approach, i.e. the classifier was trained on the data of 22 subjects and then applied to a 23rd subject by using the leave-one-out cross validation method.
Specifically, we used a MVPA combining feature selection techniques with a support vector machine (SVM). We applied the SVM implementation of WEKA software [69]. The exceptional high data dimensionality of the data was reduced by applying Information Gain feature selection [70]. This technique selects the most relevant features of the data by measuring the information gain with respect to the entropy (H) of a class before and after observing the feature:

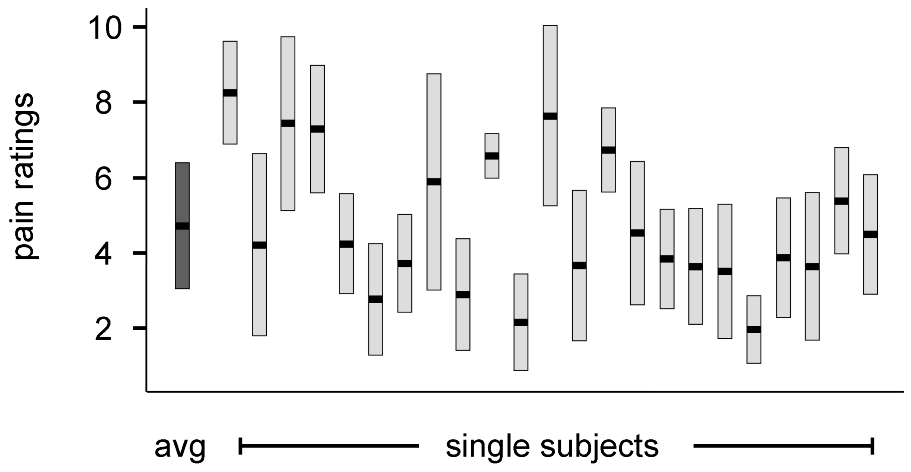
$$InfoGain(Class, Feature) = H(Class) - H(Class|Feature)$$

Within WEKA software, we applied the SVM algorithm by Platt [156] which has the capacity to find the largest margin hyperplane separating the training data. A larger margin allows for a better generalization of the object properties. Furthermore, we used a linear SVM kernel which permits a direct interpretation of the weight vector as class separating information. Hence, the support vector weights allow to directly visualize the predictive value of each feature.
Significance of classification accuracy was assessed by computing 95% confidence intervals of classification accuracy using the efficient-score method [138]. In particular, we applied the Wilson procedure with continuity correction. According to this method, classification accuracy of single trial labels (low pain vs. high pain) above 59% was considered significantly different from chance (50%). With respect to classification of an individual's pain sensitivity, accuracy above 73% was considered significantly different from chance (50%). In addition, permutation statistics were performed. To this end, each trial was randomly labeled high pain or low pain. The classifier was trained on the randomly labeled trials and the resulting classification of the individual's pain sensitivities was determined. This procedure was repeated 1000 times and the distribution of classifications was compared to the classification based on the real data. Permutation statistics were focused on selected electrodes with high classification accuracy (Cz, FCz).

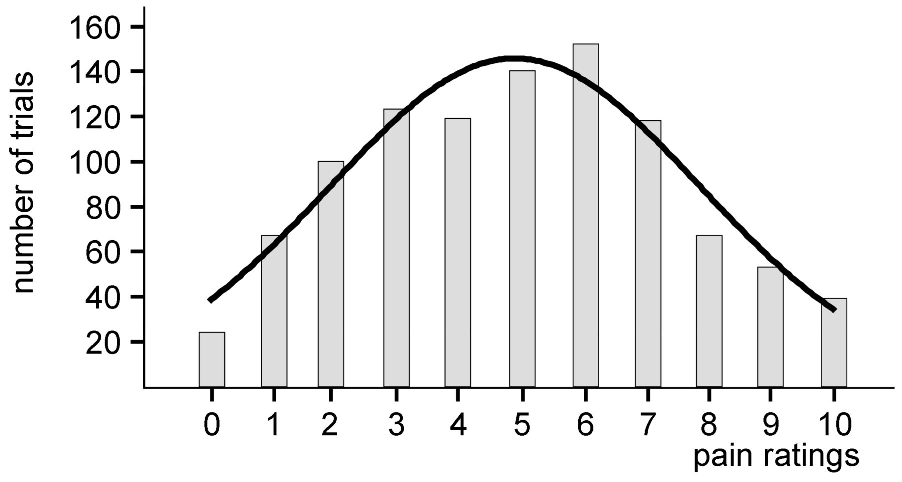
Furthermore, we compared the predictive value of the single trial TFR approach with additional approaches which do not take into account the temporal-spectral pattern of single trial responses. To this end, the SVM was also trained with the averaged TFR, the averaged evoked potential and the single trial raw data of 22 subjects. Again, the classifier was then applied to the 23rd subject. This procedure was performed for each electrode separately. Classification accuracy of these approaches to the initial approach was compared by using the non-parametric Wilcoxon signed-rank test [151]. Specifically, accuracy of all electrodes for one approach represented one sample and was compared to accuracy of all electrodes for another approach as another sample. Moreover, topographical maps were created which show the classification accuracy of each electrode across the scalp. In addition, SVM discrimination TFRs were calculated which show the support vector weights, i.e. the predictive value as a function of time and frequency.

11.3 Results

23 healthy human subjects (9 male, 14 female) with a mean age of 26 years (range 19 – 35 years) participated in the study. Informed consent was given by all subjects. The study was approved by the local ethics committee and conducted in conformity with the declaration of Helsinki.
Stimuli elicited moderately painful sensations with a group mean pain intensity of 4.9. However, pain intensity elicited by the repeated application of identical stimuli varied substantially across [42, 109] and within [160, 163] individuals (Figure 11.1). We aimed to predict the interindividual differences in the perception of identical stimuli from brain activity. We therefore labeled each subject as pain insensitive or pain sensitive based on their ratings of the painful stimuli. Specifically, subjects who rated more often ≤ 5 than > 5 were labeled pain insensitive and subjects who rated more often > 5 than ≤ 5 were labeled pain sensitive. In a next step, we aimed to predict these 'true' labels from brain activity. During the experiment, brain activity was recorded from 64 EEG electrodes. To assess the temporal-spectral pattern of different neuronal responses to pain we computed time-frequency representations (TFR) of pain-related neuronal responses. TFRs show neuronal activity as a function of time and frequency and include phase-locked and non-phase-locked neuronal responses at different frequencies. The group mean TFR at exemplary vertex electrode FCz confirms a pattern of three pain-related neu-



(A)



(B)

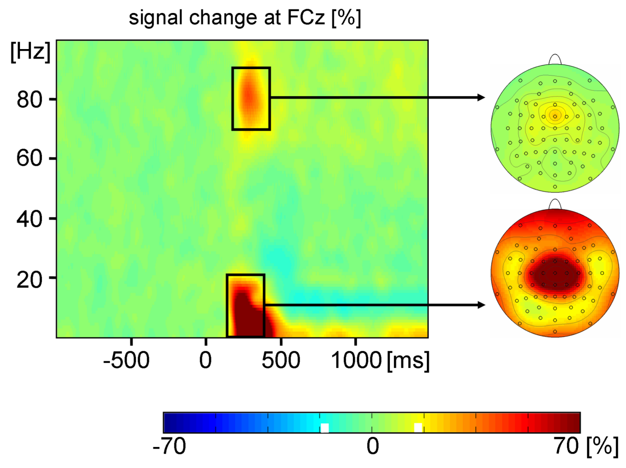
Figure 11.1: Behavioral data. (A) Individual pain ratings (light grey, mean $\pm$ standard deviations) and the mean pain rating across all subjects (black, mean $\pm$ standard deviation). Subjects with a mean pain rating of > 5 or ≤ 5 were labeled pain sensitive or pain insensitive, respectively. (B) Distribution of single trial pain ratings.

ronal responses (Figure 11.2A): evoked responses with a maximum at theta frequencies below 10 Hz and at latencies between 150 and 350 ms [55, 120], gamma responses around 80 Hz at latencies between 150 and 350 ms [63, 73], and a decrease of alpha activity around 10 Hz starting at about 500 ms after stimulus application [136, 157].

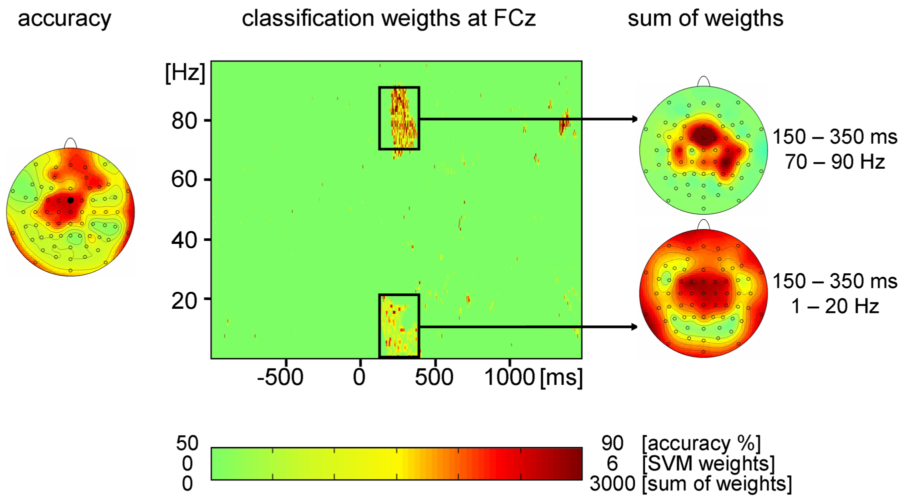
To predict an individual's pain sensitivity from pain-related EEG responses, we applied a MVPA using a support vector machine (SVM) classifier. To make use of the information of all single trials, we pursued a two-step approach. First, the classifier was trained to predict the perception of single trials. Second, we inferred an individual's sensitivity to pain from the predicted perception of the single trials. For the first step, we assigned 'true' labels to each single trial (low pain, rating ≤ 5; high pain, rating > 5) corresponding to the 'true' labels of the subjects (see above). The classifier was trained on the single trial TFRs of 22 subjects and applied to predict the single trial labels (low pain vs. high pain) of the 23rd subject. In a second step we inferred an individual's pain sensitivity from the individual proportion of low pain and high pain trials. Prediction accuracy was assessed by comparing the predicted pain sensitivity based on brain activity with the individual's 'true' pain sensitivity based on the pain ratings. The procedure was performed for each electrode. The results show that our approach allows for decoding an individual's sensitivity to pain with a maximum accuracy of 83% (19 of 23 subjects). Maximum accuracy was accomplished when using the EEG data of each of four electrodes mainly at central locations (Figure 11.2B, left; Table 11.1). Statistical testing confirmed that this accuracy was higher than chance (50%) (permutation tests at electrodes Cz and FCz, $p < 0.001$). Accuracy of classification of single trials (low pain vs. high pain) across subjects was 62% which was also significantly higher than chance (50%).

Next, we investigated which pain-related neuronal responses contribute to the classification accuracy. We therefore calculated SVM discrimination TFRs which do not show neuronal activity but the predictive value of neuronal activity as a function of time and frequency. Results show that the evoked theta and the gamma response but not the late alpha response contribute to the classification of the individual sensitivity to pain (Figure 11.2B, middle).

We finally compared our single trial TFR based analysis with approaches which do not take single trials and/or the temporal-spectral pattern of responses at different frequencies into account. We therefore averaged the



(A)



(B)

Figure 11.2: (A) Neuronal responses to painful stimuli. (left) Time-frequency representation (TFR) of neuronal activity at electrode FCz coded as percent signal change with respect to a prestimulus baseline. The group mean TFR has been averaged across trials and subjects. (right) Group mean topography of gamma and evoked responses coded as percent signal change with respect to a prestimulus baseline. (B) Decoding an individual's sensitivity to pain. (left) Topography of classification accuracy for the single trial TFR data. (middle) The SVM discrimination TFR at electrode FCz shows the classification weight, i.e. the predictive value as a function of time and frequency. (right) Topography of classification weights for the gamma response and the evoked theta response.

	single trial TFR	**mean TFR**	**single trial raw EEG**	**averaged evoked potential**
FC1	0.83/0.50/1.00	0.57/0.00/0.87	0.65/0.22/0.93	0.65/0.00/0.83
FC3	0.78/0.38/1.00	0.61/0.13/0.87	0.70/0.22/1.00	0.74/0.00/0.94
FCz	0.83/0.50/1.00	0.65/0.25/0.87	0.70/0.33/0.93	0.13/0.13/0.00
FC2	0.78/0.38/1.00	0.65/0.25/0.87	0.70/0.33/0.93	0.57/0.00/0.72
FC4	0.70/0.25/0.93	0.57/0.13/0.80	0.65/0.11/1.00	0.65/0.00/0.83
C1	0.78/0.50/0.93	0.61/0.25/0.80	0.78/0.44/1.00	0.65/0.00/0.83
C3	0.78/0.50/0.93	0.61/0.00/0.93	0.74/0.33/1.00	0.70/0.20/0.83
Cz	0.83/0.50/1.00	0.52/0.13/0.73	0.70/0.44/0.86	0.70/0.00/0.89
C2	0.74/0.38/0.93	0.61/0.38/0.73	0.74/0.44/0.93	0.57/0.00/0.72
C4	0.74/0.25/1.00	0.52/0.00/0.80	0.70/0.22/1.00	0.57/0.00/0.72
CP1	0.78/0.38/1.00	0.48/0.13/0.67	0.70/0.33/0.93	0.70/0.00/0.89
CP3	0.74/0.38/0.93	0.70/0.13/1.00	0.70/0.33/0.93	0.78/0.40/0.89
CPz	0.74/0.38/0.93	0.57/0.13/0.80	0.70/0.33/0.93	0.61/0.00/0.78
CP2	0.74/0.38/0.93	0.57/0.00/0.87	0.70/0.33/0.93	0.74/0.20/0.89
CP4	0.65/0.25/0.87	0.61/0.13/0.87	0.57/0.11/0.86	0.52/0.00/0.67

Table 11.1: Classification accuracy, sensitivity and specificity of different approaches to decode an individual's sensitivity to pain. For ease of readability, results from a selection of 15 EEG electrodes from central locations are shown.

TFRs across trials for each electrode and subject. These averaged TFRs assess the temporal-spectral pattern of neuronal responses to pain but do not include the single trial information. Classification of subjects based on this approach shows a maximum accuracy of 78% at a single electrode (Figure 11.3A). Next, we based the classification on the single trial raw EEG data which include the single trial information but do not assess the temporal-spectral pattern of neuronal responses. The results show a maximum classification accuracy of 78% at one left hemispheric electrode (Figure 11.3B). Finally, we calculated the evoked potential by averaging the EEG data across trials for each electrode and subject. The averaged evoked potential does neither assess the temporal-spectral pattern of information nor the single trial information. This approach resulted in a maximum classification accuracy of 78% at 3 left hemispheric electrodes (Figure 11.3C). Statistical comparisons

	single trial TFR	**averaged TFR**	**single trial raw EEG averaged**
TFR	z = -6.53 $p < 0.001$	-	-
single trial raw EEG	z = -6.57 $p < 0.001$	z = -4.59 $p < 0.001$	-
averaged evoked potential	z = -6.66 $p < 0.001$	z = -4.82 $p < 0.001$	z = -5.96 $p < 0.001$

Table 11.2: Statistical comparisons of different approaches to decode an individual's sensitivity to pain. Wilcoxon signed-rank tests were used to compare accuracy of the different approaches. Specifically, accuracies of all electrodes for one approach represented one sample and were compared to accuracies of all electrodes for another approach as another sample.

confirmed that our initial approach had a significantly higher predictive value than the other approaches (all $p < 0.001$, Wilcoxon signed-rank tests, Table 11.2).

In summary, our results show that the multivariate assessment of the temporal-spectral pattern of single trial EEG responses to pain allows for significant predictions of an individual's sensitivity to pain from brain activity.

11.4 Interpretation of the Results

Here, we aimed to predict an individual's sensitivity to pain from brain activity. We repeatedly applied identical painful stimuli to healthy human subjects and recorded neuronal responses to pain by using EEG. We applied a multivariate analysis to the EEG data to predict an individual's sensitivity to pain from the temporal-spectral pattern of single trial neuronal responses to pain. The results show that this approach allows for predicting an individual's pain sensitivity with an accuracy of 83%. Intriguingly, we pursued a cross-subject approach in which the prediction was based on a classifier

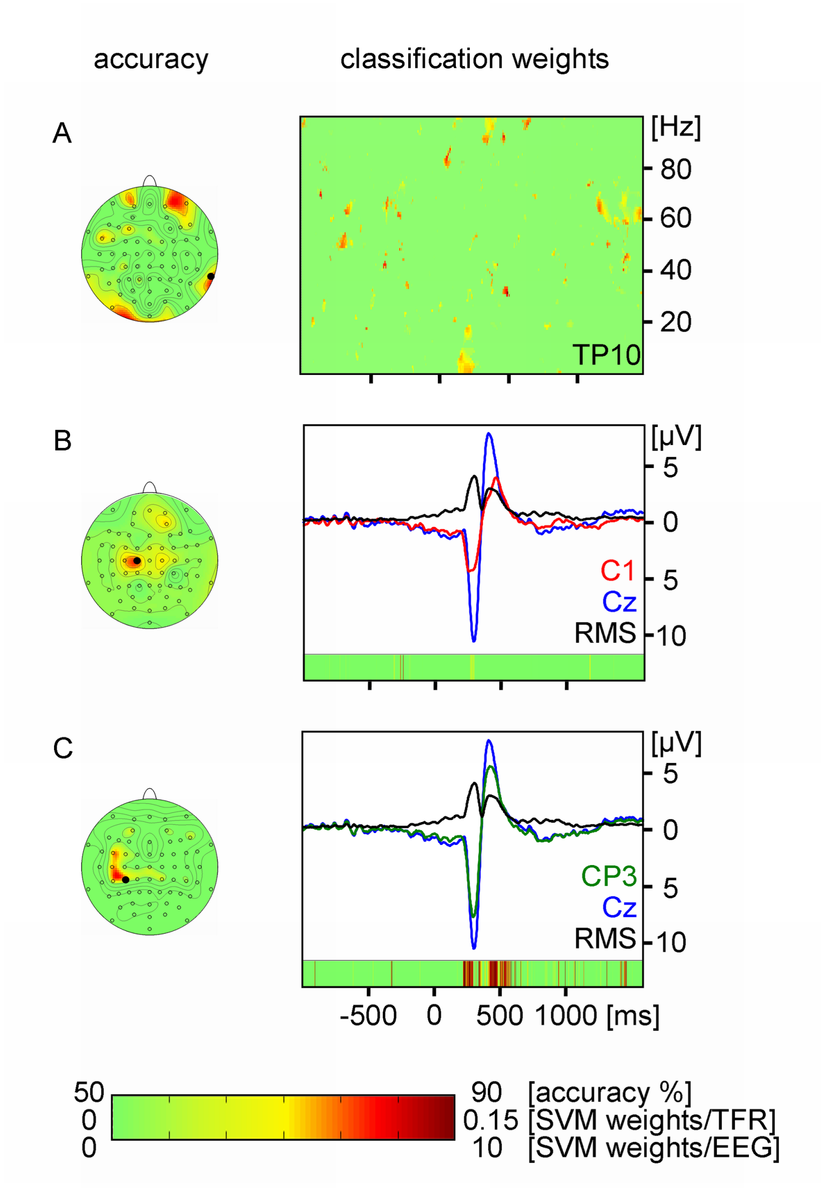


Figure 11.3: Decoding an individual's sensitivity to pain based on (A) the averaged TFR, (B) the single trial raw EEG data and (C) the averaged evoked potential. Left panels show topographies of classification accuracy. Right panels show SVM classification weights color coded as a function of time and frequency (A) and as a function of time (B,C). (A) and (C) also include time courses of signals at selected electrodes and root mean squares (RMS) of all electrodes.

trained on independent data from other individuals. Our findings may thus help to establish a much sought-after objective neuronal marker of an individual's sensitivity to pain [27, 187].
Our results corroborate a substantial interindividual variability in the perception of identical painful stimuli. Differences in sensitivity are observed in all sensory modalities but particularly apply to the perception of pain [42, 109, 139] where the individual sensitivity influences future pain experiences and responses to analgesic treatment [45, 139]. Variations in pain sensitivity may be caused at any stage in pain processing from the skin to the brain. Mechanisms contributing to interindividual differences in pain sensitivity include genetic, environmental, psychological and cognitive factors [41, 139]. However, regardless of its origin and the underlying mechanisms, perceptual variation should be finally reflected in the neural activity of the brain which ultimately determines the perception of pain. Correspondingly, an fMRI study indicated that pain-related BOLD responses in somatosensory, anterior cingulate and prefrontal cortices reflect differences in the sensitivity to pain [42]. An EEG study showed that pain sensitivity correlates with amplitudes of evoked responses [87]. Our study complements and extends these studies by revealing that different neuronal responses do not only correlate with the sensitivity to pain but that the multivariate assessment of the temporal-spectral pattern of pain-related neuronal responses allows for predicting an individual's sensitivity to pain from brain activity.
Specifically, our results show that the prediction of an individual's sensitivity to pain depends on the pattern of pain-evoked responses at around 8 Hz and pain-induced gamma oscillations at about 80 Hz. This approach yielded significantly higher classification accuracy than approaches which do not take the temporal-spectral pattern of single trial responses into account. We specifically compared our approach with an approach which is based on an individual's averaged evoked potential. As the latter approach includes only a single time series it represents virtually a univariate analysis. Moreover, our approach was also more powerful than an approach based on an individual's averaged TFR which indicates that single trials contain information which goes beyond the information of an average. Our results, thus, reveal that the cerebral representation of pain is inherently multivariate and that the temporal-spectral pattern of neuronal responses to pain provides crucial information about the perception of pain. Consequently, multivariate approaches which jointly assess temporal-spectral and/or spatial patterns of neuronal responses appear particularly promising to further the understand-

ing of the cerebral representation of pain.
Our approach builds upon recent MVPA of functional magnetic resonance imaging (fMRI) [77, 141] and electrophysiological [121] data which are intended to identify patterns of brain activity that differentiate between mental states (e.g. [51, 75, 76, 98]). A recent fMRI study used MVPA to predict three objectively different levels of thermal stimulation from the spatial pattern of brain activity [128]. Here, we go beyond that study and most other 'brain reading' applications (see [135, 158, 173] for exceptions) by applying a cross-subject approach. To this end, the classifier was trained on a group of subjects and then applied to another subject to predict the subject's sensitivity to pain. Moreover, we did not distinguish between objectively different intensities of thermal stimuli but we predicted interindividual differences in the perception of identical stimuli. In addition, we used EEG recordings to assess the temporal-spectral rather than the spatial pattern of single trial responses to pain. The cross-subject approach, the prediction of subjective differences in the perception of identical stimuli and the use of rather cheap and simple EEG recordings may represent a step further towards translation of 'brain reading' approaches into practical applications.
However, several limitations of the present approach should be noted. First, we used a simplified dichotomous model of pain sensitivity. Such a simplified categorical model appears robust and reasonable as a first step towards practical applications but does not necessarily generalize to all conditions and subjects. Second, the relationship between pain perception and neuronal responses is not fully stable. Specifically, short and constant interstimulus intervals can disrupt the relationship between amplitudes of neuronal responses and pain perception [86]). Our approach thus applies only to sufficiently long and varied interstimulus intervals (8 - 12 sec). Third, our assessment refers to the momentary sensitivity to pain which can be influenced by a broad variety of internal and external factors including skin temperature. Future studies should therefore define the experimental conditions as thoroughly as possible including assessments of skin temperature. Fourth, we used a linear support vector machine to decode pain sensitivity. This classifier is often used in 'brain reading' studies but other linear and non-linear classifiers may be equally or even more powerful (see [137, 151] for a discussion of different classifiers). Fifth, we only assessed the temporal-spectral but not the spatial pattern of responses and, as most previous MVPA of brain activity, we performed a binary classification. The integration of temporal-spectral information with spatial information assessed by a source analysis of EEG or

even by fMRI could potentially further improve the multivariate assessment of pain sensitivity. However, integration of spatial information and classification of multiple classes would be computationally highly demanding. For example, a source based approach to the EEG data even when using a large voxel size of 7 mm would multiply the data by the factor 100 as compared to the present electrode based approach.

11.5 Conclusion

In this chapter, we proposed an approach to predict an individual's sensitivity to pain by applying a multivariate analysis to EEG data. The approach is based on a very simple experiment where the data can potentially be recorded within a few minutes from a single EEG electrode. The results of the observations can be useful in determining a simple neuronal marker of pain perception which in the turn could estimate an individual's sensitivity to pain. This is especially helpful in the cases when verbal report from a person is not possible or reliable, e.g. in situations with critically ill or demented patients or patients suspected of malingering. For such patients the characterization of an individual's pain sensitivity could help to optimize general care and analgesic treatment. Our approach could thereby help to improve the prevention, diagnosis and treatment of pain with implications for health care [27, 187] and the legal system [131]. The results show that this approach allows for predicting an individual's pain sensitivity with an accuracy of 83%, which characterizes a good quality of the method.

For future work, we are going to improve the approach by using other linear and non-linear classifiers. Result improvements can be expected from the additional analysis of skin temperature. Interesting results are also expected from integration of temporal-spectral information with spatial information.

Part V

Conclusion

Chapter 12

Summary and Outlook

In the parts II, III and IV, there were considered various Data Mining algorithms, their optimization and their application in different medical researches. This chapter consists of two sections. In the first of them 12.1 the main conclusions of this thesis are summarized. Section 12.2 considers the possible directions of improvements and further development of the algorithms proposed in the thesis.

12.1 Summary

In this thesis a lot of various Data Mining methods have been taken under consideration. New algorithms based on dependencies between time series were proposed. Besides, optimization of already existing methods for specialized hardware was suggested here. Other directions of the thesis are new approaches for multidimensional time series as well as the application of both new and well-known methods in workflow describing complex medical data processing.

Part II includes optimization of such classical algorithms as similarity join, k-means and DBSCAN when applying them on GPU by means of CUDA technique. It was achieved essential computing performance gain compared to classical CPU-based algorithms. Recently, Moore's Law [133] in terms of processing speed does not hold any more for a usual CPU, but for a GPU this empirical statement is still true. To use entirely the potential of graphics cards, not only the part of time intensive computations must be done in parallel, but also specific GPU memory architecture must taken into account, which can effectively work only with big data blocks. The algorithms proposed in this part use new techniques, which shows their ability to process essentially bigger data volumes for future applications.

In Part III, propositions of mathematical models for a wide spectrum of Data Mining tasks can be found. Time series based models let efficiently aggregate information and effectively use it for approximation, classification and clustering. Under consideration here were also linear and linearizable models as well as methods for an effective calculation of models for very long time series.

In Chapter 7 there are considered mathematical models for approximation of long time series. Time series are approximated by their own models, which allow to estimate the classical euclidean distance between time series themselves. The time series representation proposed in this chapter noticeably reduce the amount of data, which are to be read from the hard disk and kept in memory, not loosing at the same time the accuracy when calculating distance between the objects. By the example of k-means it was shown that the clustering computed by the means of models is almost equivalent to the clustering built by means of original non-reduced objects and significantly better than other representations such as DFT, DWT and Chebyshev polynomials based representations.

Chapter 8 includes classification algorithm based on the models. The proposed method finds not the class objects which are similar to each other in the euclidean sense but rather the objects which better fit to each other in the context of a certain model. The latter distinguishes this method from the most of concurrent approaches. Thereby, dependencies are found, splitting the classes in the best way. Due to interpretability of the models, not only the answer can be found about the most likely object class label, but also this decision can be justified by pointing to models which influenced classification decision making. Model-based classification supplements already existing methods very well, especially the ones which do not take into account the dependencies between the dimensions of a multidimensional time series. The suggested algorithm showed very good results on different data from medicine and industry. A good classification-accuracy was achieved. As an example of application of interpretability in medicine, the regions of brain were determined which were changed by Alzheimer's and alcoholism diseases.

In Chapter 9, it was proposed a clustering algorithm based on models. The algorithm showed very good results especially on medical data. The model abstraction helped a lot to work effectively with huge data volumes. Some extensions of k-means for IKM was adapted here. The ability to interpret the clustering results gives a powerful tool for medical data analysis and finding new interesting dependencies in these data. Here it was also extended the notion of model by adding to linear ones also linearizable models. It was searched not only for linear dependencies but also for linearized more complex nonlinear dependencies. This allowed to find more accurate and closer to real world clustering. The proposed algorithms are well-scalable and noise robust. Using IKM it can be seen by means of fMRI images the brain regions which are most affected among pain disorder persons. Besides, the regions were found which are responsible for Alzheimer's disease. The approach proposed in this chapter in combination with nonlinear models also successfully determined the part of brain regions, the functions of which were changed by schizophrenia. The importance of regions, found by means of clustering, is verified by numerous medical publications.

The Part IV presents the application of different Data Mining algorithms for exploration of complex medical data. In Chapter 10 it was proposed various hierarchical cluster algorithms for diverse data analysis. By means of model-based AV-Link, the dependency between aging and Alzheimer's disease was researched. A new concept of distance between model-based clusters was introduced here. This algorithms helped to confirm the hypothesis about Alzheimer's disease as aging process. The algorithm found on the most of clinical data representations expected and interpretable hierarchies.

In Chapter 10 it was also proposed a hierarchical subspace model based clustering to search of dependencies only within a subset of dimensions composing a multidimensional (i.e. multivariate) time series. Subspace algorithm using a different number of models did not show the expected results. However, the subspace algorithm based on the different model length gave very good results and found clear and interpretable hierarchies on real-world datasets. Besides, the method showed noise robustness.

Chapter 11 is dedicated to the application of the classical algorithms such as dimension reduction, classification and voting to analyze the pain reaction by healthy controls. A test for sensitivity to pain was proposed here as well. By means of InfoGain and SVM techniques, an analysis of EEG records of laser stimulation was carried out. New effective methods for processing of big time-frequency representations were proposed here. The obtained brain regions which are responsible for pain are approved by many researches carried out in this field. The approach proposed here is simple enough to be clinically applied and can be used to diagnose chronic phantom pains or just hypersensitivity to pain.

12.2 Outlook

In the last section of this thesis, it will be considered the main directions of development and improvement of the algorithms proposed above.

The algorithms proposed in the Part II are based on the NVIDIA's proprietary technology CUDA. Nowadays, also the platforms from competitors are being developed. In the future, we plan to adapt our algorithms for open cross-platform programming languages such as OpenCL. To achieve a good scalability of modern computing systems it is necessary to adapt the algorithms to graphic processors and graphic memory.

Model-based dependency analysis proposed in Part III gave very good results in many Data Mining algorithms. The model search in these algorithms represents a serious task. The main complexity here is finding of the subset of dependent dimensions. The solution of this problem takes the most of time. Getting approved and accelerated the model search, we obtain more accurate and flexible Data Mining tool. The proposed heuristic BIC reaches its limit when building dendrograms. Dealing with a large number of dimensions, BIC does not achieve optimal results. Improving estimation quality of selected dimensions is the major task for wide dissemination of these algorithms. Model-based approach should show good results in outlier search. Objects with a too big error in all models are certainly outliers. In the thesis we touched on the question of building object hierarchies based on models. These algorithms encounter problems caused by inaccuracy in found models and, as consequence, by a strong inaccuracy in the resulting hierarchy. In the future we want to refine these algorithms by making the model search more accurate, determining more flexible and general clustering models and getting rid of data dependent parameters in the algorithms.

In this thesis in Part IV proposed algorithms for classification and clustering of medical data showed very good results. We plan to optimize these algorithms for dealing with large amount of data and in the future introduce them into clinical diagnostics. For this, the usage of graphic processors with solutions from Part II will be very helpful. We are going to create large data bases with records of various diseases for machine-supported diagnostics. The purpose of researches in this thesis is creation of accurate and fast system for diagnostication of known diseases and searching for their new subtypes.

Chapter 13

Appendix

IC DMN	peak voxel x, y, z	Anatomy (AAL)	Cluster size (voxel)	t peak voxel	p FWE cluster level
1	3, -67, 31	right precuneus	5958	41.70	0.001
2	-3, -52, 22	left posterior cingulum	2669	55.35	0.001
3	0, 50, 4	left anterior cingulum	3572	44.69	0.001
4	18, -25, -17	right parahippocampus	3274	34.54	0.001
Attentional					
1	-45, -61, 40	left angular gyrus	1300	37.03	0.001
2	45, -58, 40	right angular gyrus	4227	34.23	0.001
3	57, -46, 19	right superior temporal cortex	6398	33.86	0.001
4	-36, 14, -11	left insula	980	35.43	0.001
1	-45, 20, -5	left inferior orbital frontal cortex	7972	39.77	0.001

2	-45, 20, 28	left inferior orbital fontal gyrus (triangularis)	4493	36.52	0.001
3	-39, 47,13	left middle frontal gyrus	12001	35.38	0.001
Visual					
1	0, -79, 4	left lingual gyrus	2610	29.28	0.001
2	3, -73, 4	right lingual gyrus	3286	41.96	0.001
3	9, -67, -5	right lingual gyrus	2798	42.95	0.001
4	-27, -8, 22	left middle occipital cortex	9354	30.70	0.001
Auditory Network	-51, -25, 10	left temporal superior cortex	2363	33.79	0.001
Sensory-motor					
1	-51, -10, 31	left postcentral gyrus	1511	34.09	0.001
2	45, -34, 46	right parietal inferior cortex	12525	29.11	0.001
3	-48, -34, 40	left parietal inferior cortex	4382	30.73	0.001
4	0, 20, 43	left superior motor area	4248	38.01	0.001
5	6, -22, 55	right superior motor area	3920	41.86	0.001
Basal Ganglia Network	-24, 2, -8	left putamen	1982	34.87	0.001

Table 13.1: Peak voxels, coordinates, anatomical labeling and cluster size of the identified 22 intrinsic networks. AAL= anatomical automatic labeling; DMN= default mode network; FWE= family wise error corrected; ROI= region of interest.

IC	ROI coordinates peak voxel, x y , z			Anatomy (AAL)	Cluster size (voxels)	t peak voxel	p FWE cluster level
DMN							
1	3	-67	31	right precuneus	5958	41.70	0.001
2	-3	-52	22	left posterior cingulum	2669	55.35	0.001
	48	-61	34	right angular gyrus	2669	12.28	0.001
	-45	-64	34	left angular gyrus	178	14.71	0.001
3	0	50	4	left anterior cingulum	3572	44.69	0.001
4	18	-25	-17	right parahippocampus	3274	34.54	0.001
	-24	-16	-14	left hippocampus	3274	30.97	0.001
Attentional							
1	-45	-61	40	left angular gyrus	1300	37.03	0.001
	-21	32	43	left superior frontal gyrus	2865	28.42	0.001
	54	-1	4	right rolandic operculum	1488	18.67	0.001
	51	-64	34	right angular gyrus	1488	13.10	0.001
	-6	-61	37	left precuneus	828	18.00	0.001
	39	-70	-41	right cerebellum	420	17.49	0.001
	21	32	55	right superior frontal gyrus	239	8.47	0.001
2	45	-58	40	right angular gyrus	4227	34.23	0.001
	-45	-58	40	left angular gyrus	5854	26.06	0.001
	42	20	49	right middle frontal cortex	5854	23.14	0.001
	-42	50	4	left middle frontal cortex	547	17.71	0.001
	0	-52	-35	vermis 9 (cerebellum)	403	8.18	0.001
3	57	-46	19	right superior temporal cortex	6398	33.86	0.001

	-3	-52	49	left precuneus	6398	25.89	0.001
	-54	-49	13	left middle temporal	1830	27.03	0.001
	-48	-10	31	left postcentral	500	8.49	0.001
4	-36	14	-11	left insula	980	35.43	0.001
	39	17	-8	right insula	890	30.93	0.001
	3	38	28	right anterior cingulum	462	10.46	0.001
Frontal							
1	-45	20	-5	left inferior orbital frontal cortex	7972	39.77	0.001
	51	23	-5	right inferior orbital frontal	1248	29.79	0.001
2	-45	20	28	left inferior orbital fontal gyrus (triangularis)	4493	36.52	0.001
	42	11	31	right frontal inferior operculum	2013	31.53	0.001
	-33	-58	46	left inferior parietal cortex	845	21.83	0.001
	36	-61	49	right superior parietal cortex	325	13.51	0.001
3	-39	47	13	left middle frontal gyrus	12001	35.38	0.001
	30	56	7	right superior frontal	12001	26.19	0.001
Visual							
1	0	-79	4	left lingual gyrus	2610	29.28	0.001
	3	-55	40	right precuneus	854	10.37	0.001
2	3	-73	4	right lingual gyrus	3286	41.96	0.001
3	9	-67	-5	right lingual gyrus	2798	42.95	0.001
4	-27	-82	22	left middle occipital cortex	9354	30.70	0.001
	30	-76	25	right middle occipital	9354	29.35	0.001
Auditory	-51	-25	10	left temporal superior cortex	2363	33.79	0.001

	63	-25	13	right temporal superior cortex	1829	32.60	0.001
Sensory-motor							
1	-51	-10	31	left postcentral gyrus	1511	34.09	0.001
	54	-7	31	right postcentral gyrus	1624	33.83	0.001
	15	-64	-20	right cerebellum	495	17.84	0.001
	-18	-64	-23	left cerebellum	495	16.93	0.001
2	45	-34	46	right parietal inferior cortex	12525	29.11	0.001
3	-48	-34	40	left parietal inferior cortex	4382	30.73	0.001
	51	-28	43	right postcentral gyrus	1661	29.26	0.001
	54	-58	-5	right inferior temporal	1142	17.27	0.001
	57	11	28	right frontal inferior operculum	707	9.22	0.001
4	0	20	43	left superior motor area	4248	38.01	0.001
5	6	-22	55	right superior motor area	3920	41.86	0.001
Basal Ganglia	-24	2	8	left putamen	1982	34.87	0.001
	21	8	4	right putamen	1660	32.62	0.001

Table 13.2: Peak voxels, coordinates, anatomical labeling and cluster size of the 55 selected peak voxels from the previously identified 22 intrinsic networks. AAL= anatomical automatic labeling; DMN= default mode network; FWE= family wise error corrected; ROI= region of interest.

Bibliography

[1] Illustration from american health assistance foundation. `http://www.alzhyme.com/images/`. Accessed: 2013-09-30.

[2] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[3] E. Achtert, C. Böhm, P. Kröger, and A. Zimek. Mining hierarchies of correlation clusters. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 119–128. IEEE, 2006.

[4] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek. On exploring complex relationships of correlation clusters. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 7–7. IEEE, 2007.

[5] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. In *In Proc 1999 ACM-SIGMOD Int. Conf. Management of Data (SIFMOND'99)*, volume 28, pages 61–72. ACM, 1999.

[6] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *ACM SIGMOD Record*, volume 29, pages 70–81. ACM, 2000.

[7] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, pages 69–84, 1993.

[8] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. In SIGMOND Record ACM Special Interest Group on Management of Data, 1998.

[9] E. A. Allen, E. B. Erhardt, E. Damaraju, W. Gruner, J. M. Segall, R. F. Silva, M. Havlicek, S. Rachakonda, J. Fries, R. Kalyanam, et al. A baseline for the multivariate comparison of resting-state networks. *Frontiers in systems neuroscience*, 5, 2011.

[10] O. Alter, P. Brown, and D. Botstein. Generalized singular value decomposition for comparative analysis of genome-scale expression data sets of two different organisms. *Proc Natl Acad Sci U S A*, 100(6):3351–6, 2003.

[11] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[12] J. Anderson. *Cognitive Psychology and its Implications, Ch. 6, Human Memory: Encoding and Storage*. Worth Publishers, 2004.

[13] A. V. Apkarian, M. C. Bushnell, R. D. Treede, and J. K. Zubieta. Human brain mechanisms of pain perception and regulation in health and disease. *Eur J Pain*, 2005.

[14] D. Arthur, B. Manthey, and H. Röglin. Smoothed analysis of the k-means method. *Journal of the ACM (JACM)*, 58(5):19:1–19:31, Oct. 2011.

[15] K. Backhaus, B. Erichson, W. Plinke, and R. Weiber. Multivariate Analysemethoden–eine anwendungsorientierte Einführung, 10., neu bearb. und erw (in German). *Aufl., Berlin/Heidelberg/New York*, 2003.

[16] Z. Bar-Joseph, G. K. Gerber, D. K. Gifford, T. S. Jaakkola, and I. Simon. Continuous representations of time-series gene expression data. *Journal of Computational Biology*, 10(3-4):341–356, 2003.

[17] R. J. Bateman, C. Xiong, T. L. Benzinger, A. M. Fagan, A. Goate, N. C. Fox, D. S. Marcus, N. J. Cairns, X. Xie, T. M. Blazey, et al. Clinical and biomarker changes in dominantly inherited alzheimer's disease. *New England Journal of Medicine*, 367(9):795–804, 2012.

[18] T. Bekhuis. Pain disorder. `http://www.minddisorders.com/Ob-Ps/Pain-disorder.html`, 2013.

[19] F. Bermejo-Pareja, J. Benito-León, S. Vega, M. Medrano, and G. Román. Incidence and subtypes of dementia in three elderly populations of central spain. *Journal of the Neurological Sciences*, 264(1–2):63 – 72, 2008.

[20] D. J. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, and B.-Y. Yang. Ecm on graphics cards. In *EUROCRYPT*, pages 483–501, 2009.

[21] K. Blennow, M. de Leon, and H. Zetterberg. Alzheimer's disease. *The Lancet*, 368(9533), 2006.

[22] C. Böhm, B. Braunmüller, M. M. Breunig, and H.-P. Kriegel. High performance clustering based on the similarity join. In *CIKM*, pages 298–305, 2000.

[23] C. Böhm, K. Kailing, P. Kröger, and A. Zimek. Computing clusters of correlation connected objects. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 455–466. ACM, 2004.

[24] C. Böhm, L. Läer, C. Plant, and A. Zherdin. Model-based classification of data with time series-valued attributes. In *BTW*, pages 287–296, 2009.

[25] C. Böhm, R. Noll, C. Plant, B. Wackersreuther, and A. Zherdin. Data mining using graphics processing units. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems I*, pages 63–90. Springer, 2009.

[26] C. Böhm, R. Noll, C. Plant, and A. Zherdin. Indexsupported similarity join on graphics processors. In *BTW*, pages 57–66, 2009.

[27] D. Borsook, S. Sava, and L. Becerra. The pain imaging revolution: Advancing pain into the 21st century. *The Neuroscientist*, 16(2):171–185, 2010.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD Conference*, pages 93–104, 2000.

[29] P. Brockwell and R. Davis. *Introduction to Time Series and Forecasting*. Number Bd. 1 in Introduction to Time Series and Forecasting. Springer, 2002.

[30] A. S. Brown. Prenatal infection as a risk factor for schizophrenia. *Schizophrenia Bulletin*, 32(2):200–202, 2006.

[31] R. L. Buckner, J. Sepulcre, T. Talukdar, F. M. Krienen, H. Liu, T. Hedden, J. R. Andrews-Hanna, R. A. Sperling, and K. A. Johnson. Cortical hubs revealed by intrinsic functional connectivity: mapping, assessment of stability, and relation to alzheimer's disease. *The Journal of Neuroscience*, 29(6):1860–1873, 2009.

[32] R. L. Buckner, A. Z. Snyder, A. L. Sanders, M. E. Raichle, J. Morris, et al. Functional brain imaging of young, nondemented, and demented older adults. *Journal of Cognitive Neuroscience*, 12(Supplement 2):24–34, 2000.

[33] R. L. Buckner, A. Z. Snyder, B. J. Shannon, G. LaRossa, R. Sachs, A. F. Fotenos, Y. I. Sheline, W. E. Klunk, C. A. Mathis, J. C. Morris, et al. Molecular, structural, and functional characterization of alzheimer's disease: evidence for a relationship between default activity, amyloid, and memory. *The Journal of Neuroscience*, 25(34):7709–7717, 2005.

[34] L. K. C. Li and B. Prabhakaran. Feature selection for classification of variable length multiattribute motions. In V. A. Petrushin and L. Khan, editors, *Multimedia Data Mining and Knowledge Discovery*. Springer, 2007.

[35] Y. Cai and R. T. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD Conference*, pages 599–610, 2004.

[36] F. Cao, A. K. H. Tung, and A. Zhou. Scalable clustering using graphics processors. In *WAIM*, pages 372–384, 2006.

[37] B. C. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *ICML*, pages 104–111, 2008.

[38] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999.

[39] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. 1996.

[40] Y. Chen, G. Dong, S. Member, J. Han, S. Member, B. W. Wah, and J. Wang. Regression cubes with lossless compression and aggregation. *IEEE Trans. Knowledge and Data Engineering*, 18:2006.

[41] R. C. Coghill. Individual differences in the subjective experience of pain: New insights into mechanisms and models. *Headache: The Journal of Head and Face Pain*, 50(9):1531–1535, 2010.

[42] R. C. Coghill, J. G. McHaffie, and Y.-F. Yen. Neural correlates of interindividual differences in the subjective experience of pain. *Proceedings of the National Academy of Sciences*, 100(14):8538–8542, 2003.

[43] T. Crow. Schizophrenia as failure of hemispheric dominance for language. *Trends in Neurosciences*, 20(8):339 – 343, 1997.

[44] B. Dom. An information-theoretic external cluster-validity measure. Technical Report RJ 10219, IBM Research Division, May 2001.

[45] R. R. Edwards. Individual differences in endogenous pain modulation as a risk factor for chronic pain. *Neurology*, 65(3):437–443, 2005.

[46] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[47] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD Conference*, pages 419–429, 1994.

[48] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *KDD*, pages 82–88, 1996.

[49] G. Fein and M. Chang. Smaller feedback ern amplitudes during the bart are associated with a greater family history density of alcohol problems in treatment-naive alcoholics. *Drug and alcohol dependence*, 92(1-3):141–8, Jan 2008.

[50] R. Fillingim. Sex, gender, and pain: Women and men really are different. *Current Review of Pain*, 4(1):24–30, 2000.

[51] E. Formisano, F. De Martino, M. Bonte, and R. Goebel. "who" is saying "what"? brain-based decoding of human voice and speech. *Science*, 322(5903):970–973, 2008.

[52] E. Formisano and R. Goebel. Tracking cognitive processes with functional mri mental chronometry. *Current Opinion in Neurobiology*, 13(2):174–181, 2003.

[53] M. D. Fox and M. E. Raichle. Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging. *Nat Rev Neurosci*, 8(9):700–711, 2007.

[54] D. R. Galasko, F. A. Schmitt, S. Jin, J. Saxton, D. Bennett, M. Sano, and S. H. Ferris. Detailed assessment of cognition and activities of daily living in moderate to severe alzheimer's disease. *Neurobiology of Aging*, 21:168, 2000.

[55] L. Garcia-Larrea, M. Frot, and M. Valeriani. Brain generators of laser-evoked potentials: from dipoles to functional significance. *Neurophysiologie Clinique/Clinical Neurophysiology*, 33(6):279 – 292, 2003.

[56] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *KDD*, pages 81–90, 2000.

[57] E. I. George. The variable selection problem. *J. Amer. Statist. Assoc*, 95:1304–1308, 2000.

[58] J. O. Goh. Functional dedifferentiation and altered connectivity in older adults: neural accounts of cognitive aging. *Aging and disease*, 2(1):30, 2011.

[59] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha. Gputerasort: high performance graphics co-processor sorting for large database management. In *SIGMOD Conference*, pages 325–336, 2006.

[60] N. K. Govindaraju, B. Lloyd, W. Wang, M. C. Lin, and D. Manocha. Fast computation of database operations using graphics processors. In *SIGMOD Conference*, pages 215–226, 2004.

[61] C. L. Grady. Cognitive neuroscience of aging. *Annals of the New York Academy of Sciences*, 1124(1):127–144, 2008.

[62] C. L. Grady, A. R. McIntosh, and F. I. Craik. Age-related differences in the functional connectivity of the hippocampus during memory encoding. *Hippocampus*, 13(5):572–586, 2003.

[63] J. Gross, A. Schnitzler, L. Timmermann, and M. Ploner. Gamma oscillations in human primary somatosensory cortex reflect pain perception. *PLoS Biol*, 5(5):e133, 04 2007.

[64] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD Conference*, pages 73–84, 1998.

[65] H. Gündel, M. Valet, C. Sorg, D. Huber, C. Zimmer, T. Sprenger, and T. Tölle. Altered cerebral response to noxious heat stimulation in patients with somatoform pain disorder. *Pain.*, 137:413–421, Nov 2007.

[66] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[67] A. Hafkemeijer, J. van der Grond, and S. A. Rombouts. Imaging the default mode network in aging and dementia. *Biochimica et Biophysica Acta (BBA) - Molecular Basis of Disease*, 1822(3):431 – 441, 2012.

[68] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2):107–145, 2001.

[69] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[70] M. A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Trans. on Knowl. and Data Eng.*, 15(6):1437–1447, Nov. 2003.

[71] J. Hamilton, G. Chen, M. Thomason, M. Schwartz, and I. Gotlib. Investigating neural primacy in major depressive disorder: multivariate

granger causality analysis of resting-state fmri time-series data. *Mol Psychiatry*, 16(7):763–72, 2011.

[72] J. Han and M. Kamber. *Data mining: concepts and techniques.* The Morgan Kaufmann Series In Data Management Systems. Morgan Kaufmann, 2000.

[73] M. Hauck, J. Lorenz, and A. K. Engel. Attention to painful stimulation enhances γ-band activity and synchronization in human sensorimotor cortex. *The Journal of Neuroscience*, 27(35):9270–9277, 2007.

[74] O. Hauk. Introduction to eeg and meg. `http://imaging.mrc-cbu.cam.ac.uk/meg/IntroEEGMEG`, 2013.

[75] J. V. Haxby, M. I. Gobbini, M. L. Furey, A. Ishai, J. L. Schouten, and P. Pietrini. Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science*, 293(5539):2425–2430, 2001.

[76] J.-D. Haynes and G. Rees. Predicting the stream of consciousness from activity in human visual cortex. *Current Biology*, 15(14):1301–1307, 2005.

[77] J.-D. Haynes and G. Rees. Decoding mental states from brain activity in humans. *Nature Reviews Neuroscience*, 7(7):523–534, 2006.

[78] B. He, K. Yang, R. Fang, M. Lu, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational joins on graphics processors. In *SIGMOD*, pages 511–524, 2008.

[79] L. E. Hebert, P. A. Scherr, J. L. Bienias, D. A. Bennett, and D. A. Evans. Alzheimer disease in the us population: prevalence estimates using the 2000 census. *Archives of neurology*, 60(8):1119, 2003.

[80] H. Heekeren, S. Marrett, and L. Ungerleider. The neural systems that mediate human perceptual decision making. *Nat. Rev. Neurosci.*, 9(6):467–79, Jun 2008.

[81] K. Herrup. Reimagining alzheimer's disease—an age-based hypothesis. *The Journal of Neuroscience*, 30(50):16755–16762, 2010.

[82] A. Holden and W. Winlow. *The Neurobiology of Pain: Symposium of the Northern Neurobiology Group, Held at Leeds on 18 April, 1983.* Studies in Neuroscience. Manchester University Press, 1984.

[83] S. A. Huettel, A. W. Song, and G. McCarthy. *Functional Magnetic Resonance Imaging.* Sinauer Associates Inc., 2009.

[84] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent Component Analysis.* Wiley, New York, 2001.

[85] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[86] G. Iannetti, N. P. Hughes, M. C. Lee, and A. Mouraux. Determinants of laser-evoked eeg responses: pain perception or stimulus saliency? *Journal of neurophysiology*, 100(2):815–828, 2008.

[87] G. Iannetti, L. Zambreanu, G. Cruccu, and I. Tracey. Operculoinsular cortex encodes pain intensity at the earliest stages of cortical processing as indicated by amplitude of laser-evoked potentials in humans. *Neuroscience*, 131(1):199–208, 2005.

[88] C. R. Jack Jr, D. S. Knopman, W. J. Jagust, R. C. Petersen, M. W. Weiner, P. S. Aisen, L. M. Shaw, P. Vemuri, H. J. Wiste, S. D. Weigand, et al. Tracking pathophysiological processes in alzheimer's disease: an updated hypothetical model of dynamic biomarkers. *The Lancet Neurology*, 12(2):207–216, 2013.

[89] W. J. Jagust and E. C. Mormino. Lifespan brain activity, β-amyloid, and alzheimer's disease. *Trends in cognitive sciences*, 15(11):520–526, 2011.

[90] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[91] K. Jefferies and N. Agrawal. Early-onset dementia. *Advances in Psychiatric Treatment*, 15(5):380–388, 2009.

[92] T.-P. Jung, S. Makeig, M. Westerfield, J. Townsend, E. Courchesne, and T. J. Sejnowski. Removal of eye activity artifacts from visual event-related potentials in normal and clinical subjects. *Clinical Neurophysiology*, 111(10):1745–1758, 2000.

[93] M. W. Kadous. Learning comprehensible descriptions of multivariate time series. In *ICML*, pages 454–463, 1999.

[94] M. W. Kadous and C. Sammut. Classification of multivariate time series and structured data using constructive induction. *Mach. Learn.*, 58(2-3):179–216, 2005.

[95] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *Proc. SDM*, volume 4, 2004.

[96] G. J. Katz and J. T. Kider. All-pairs shortest-paths for large graphs on the gpu. In *Graphics Hardware*, pages 47–55, 2008.

[97] L. Kaufman and P. J. Rousseeuw. *Finding groups in data an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[98] K. N. Kay, T. Naselaris, R. J. Prenger, and J. L. Gallant. Identifying natural images from human brain activity. *Nature*, 452(7185):352–355, 2008.

[99] A. Kehagias and V. Petridis. Predictive modular neural networks for time series classification. *Neural Networks*, 10(1):31–49, 1997.

[100] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[101] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, pages 151–162, 2001.

[102] E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, pages 206–215, 2004.

[103] M. Kitsuregawa, L. Harada, and M. Takagi. Join strategies on kd-tree indexed relations. In *ICDE*, pages 85–93, 1989.

[104] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *SSD*, pages 47–66, 1995.

[105] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD Conference*, pages 289–300, 1997.

[106] H.-P. Kriegel, P. Kröger, A. Pryakhin, M. Renz, and A. Zherdin. Approximate clustering of time series using compact model-based descriptions. In *DASFAA*, pages 364–379, 2008.

[107] M. L. Kringelbach. The human orbitofrontal cortex: linking reward to hedonic experience. *Nature Reviews Neuroscience*, 6:691–702, 2005.

[108] M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using passing—through regions. *Pattern Recogn. Lett.*, 20(11-13):1103–1111, 1999.

[109] L. H. Lanier. Variability in the pain threshold. *Science*, 1943.

[110] D. T. Larose. *Data Mining Methods and Models.* John Wiley & Sons, 2006.

[111] D. Lehmann and W. Skrandies. Reference-free identification of components of checkerboard-evoked multichannel potential fields. *Electroencephalography and clinical neurophysiology*, 48(6):609–621, 1980.

[112] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. Str: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506, 1997.

[113] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: A partition-based method for similarity joins. *Proceedings of the VLDB Endowment*, 5(3):253–264, 2011.

[114] T. W. Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[115] M. D. Lieberman, J. Sankaranarayanan, and H. Samet. A fast similarity join algorithm using graphics processing units. In *ICDE*, pages 1111–1120, 2008.

[116] J. Lin and E. Keogh. Clustering of streaming time series is meaningless. In *In Proceedings of the 8th ACM SIGMOD workshop on Research issues in*, pages 56–65. ACM Press, 2003.

[117] J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *In EDBT*, pages 106–122, 2004.

[118] J. Lin, M. Vlachos, E. J. Keogh, D. Gunopulos, J.-W. Liu, S.-J. Yu, and J.-J. Le. A mpaa-based iterative clustering algorithm augmented by nearest neighbors search for time-series data streams. In *PAKDD*, pages 333–342, 2005.

[119] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig. Molecular dynamics simulations on commodity gpus with cuda. In *HiPC*, pages 185–196, 2007.

[120] J. Lorenz and L. Garcia-Larrea. Contribution of attentional and cognitive factors to laser evoked brain potentials. *Neurophysiologie Clinique/Clinical Neurophysiology*, 33(6):293–301, 2003.

[121] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, B. Arnaldi, et al. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4, 2007.

[122] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[123] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[124] W. Maier, D. Lichtermann, M. Rietschel, T. Held, P. Falkai, M. Wagner, and S. Schwab. Genetik Schizophrener Störungen Neuere Konzepte und Befunde. *Der Nervenarzt*, 70(11):955–969, 1999.

[125] J. Malmivuo and R. Plonsey. *Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields.* Oxford University Press, 1995.

[126] S. Manavski and G. Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC Bioinformatics*, 9, 2008.

[127] S. Manganaris. Learning to classify sensor data. Technical report, Vanderbilt University, 1995.

[128] A. Marquand, M. Howard, M. Brammer, C. Chu, S. Coen, and J. Mourão-Miranda. Quantitative prediction of subjective pain intensity from whole-brain fmri data using gaussian processes. *Neuroimage*, 49(3):2178–2189, 2010.

[129] A. R. McIntosh and B. Misic. Multivariate statistical analyses for neuroimaging data. *Annual Review of Psychology*, 64(1):499–525, 2013. PMID: 22804773.

[130] M. Meila. The uniqueness of a good optimum for k-means. In *ICML*, pages 625–632, 2006.

[131] G. Miller. Brain scans of pain raise questions for the law. *Science*, 323(5911):195–195, 2009.

[132] P. K. Mölsä, R. J. Marttila, and U. K. Rinne. Survival and cause of death in alzheimer's disease and multi-infarct dementia. *Acta Neurol Scand*, 74(2):103–7, 1986.

[133] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[134] F. Mörchen. Time series feature extraction for data mining using dwt and dft, 2003.

[135] J. Mourão-Miranda, A. L. Bokde, C. Born, H. Hampel, and M. Stetter. Classifying brain states and determining the discriminating activation patterns: support vector machine on functional mri data. *Neuroimage*, 28(4):980–995, 2005.

[136] A. Mouraux, J. Guerit, and L. Plaghki. Non-phase locked electroencephalogram (eeg) responses to co2 laser skin stimulations may reflect central interactions between a partial differential -and c-fibre afferent volleys. *Clinical neurophysiology*, 114(4):710–722, 2003.

[137] M. Mur, P. A. Bandettini, and N. Kriegeskorte. Revealing representational content with pattern-information fmri—an introductory guide. *Social cognitive and affective neuroscience*, 4(1):101–109, 2009.

[138] R. G. Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in medicine*, 17(8):857–872, 1998.

[139] C. S. Nielsen, R. Staud, and D. D. Price. Individual differences in pain sensitivity: measurement, causation, and consequences. *The journal of pain*, 10(3):231–237, 2009.

[140] M. Noll-Hussong, A. Otti, L. Laeer, A. Wohlschlaeger, C. Zimmer, C. Lahmann, P. Henningsen, T. Toelle, and H. Guendel. Aftermath of sexual abuse history on adult patients suffering from chronic functional pain syndromes: an fmri pilot study. *Journal of psychosomatic research*, 68(5):483–487, 2010.

[141] K. A. Norman, S. M. Polyn, G. J. Detre, and J. V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fmri data. *Trends in cognitive sciences*, 10(9):424–430, 2006.

[142] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. In *Proc. ACM SIGKDD*, pages 322–326, 1999.

[143] T. Oates, L. Firoiu, and P. R. Cohen. Clustering time series with hidden markov models and dynamic time warping. In *In Proceedings of the IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pages 17–21, 1999.

[144] L. Owsley, L. Atlas, and G. Bernard. Automatic clustering of vector time-series for manufacturing machine monitoring. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 4:3393, 1997.

[145] R. Palaniappan and D. P. Mandic. Eeg based biometric framework for automatic identity verification. *VLSI Signal Processing*, 49(2):243–250, 2007.

[146] R. Palaniappan and P. Raveendran. Single trial vep extraction using digital filter. *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, pages 249–252, 2001.

[147] C. Pamminger. *Bayesian Clustering of Categorical Time Series: An Approach Using Finite Mixtures of Markov Chain Models*. VDM-Verlag Dr. Müller, 2008.

[148] C.-h. Park, M.-H. Boudrias, H. Rossiter, and N. S. Ward. Age-related changes in the topological architecture of the brain during hand grip. *Neurobiology of aging*, 33(4):833–e27, 2012.

[149] L. Pasquini, A. Tonch, C. Plant, A. Zherdin, M. Ortner, A. Kurz, H. Förstl, C. Zimmer, T. Grimmer, A. M. Wohlschläger, V. Riedl, and C. Sorg. Intrinsic brain activity of cognitively normal older persons resembles more that of patients both with and at risk for alzheimer's disease than that of healthy younger persons. *Brain Connectivity*, 4(5):323–336, 2014.

[150] W. D. Penny, K. J. Friston, J. T. Ashburner, S. J. Kiebel, and T. E. Nichols, editors. *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. 2007.

[151] F. Pereira, T. Mitchell, and M. Botvinick. Machine learning classifiers and fmri: a tutorial overview. *Neuroimage*, 45(1):S199–S209, 2009.

[152] M. Pierre Beaulieu and I. A. for the Study of Pain. *Pharmacology of Pain*. IASP Press, International Association for the Study of Pain, 2010.

[153] C. Plant, C. Böhm, B. Tilg, and C. Baumgartner. Enhancing instance-based classification with local density: a new algorithm for classifying unbalanced biomedical data. *Bioinformatics*, 22(8):981–988, 2006.

[154] C. Plant, A. M. Wohlschlager, and A. Zherdin. Interaction-based clustering of multivariate time series. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 914–919. IEEE, 2009.

[155] C. Plant, A. Zherdin, C. Sorg, A. Meyer-Baese, and A. M. Wohlschläger. Mining interaction patterns among brain regions by clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 26(9):2237–2249, 2014.

[156] J. Platt et al. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods—support vector learning*, 3, 1999.

[157] M. Ploner, J. Gross, L. Timmermann, B. Pollok, and A. Schnitzler. Pain suppresses spontaneous brain rhythms. *Cerebral cortex*, 16(4):537–540, 2006.

[158] R. A. Poldrack, Y. O. Halchenko, and S. J. Hanson. Decoding the large-scale structure of brain function by classifying mental states across individuals. *Psychological Science*, 20(11):1364–1372, 2009.

[159] T. Polvikoski, R. Sulkava, M. Haltia, K. Kainulainen, A. Vuorio, A. Verkkoniemi, L. Niinistö, P. Halonen, and K. Kontula. Apolipoprotein e, dementia, and cortical deposition of beta-amyloid protein. *New England Journal of Medicine*, 333(19):1242–1248, 1995. PMID: 7566000.

[160] R. L. Quiton and J. D. Greenspan. Across-and within-session variability of ratings of painful contact heat stimuli. *Pain*, 137(2):245–256, 2008.

[161] L. Rabiner and B.-H. Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[162] C. A. Ratanamahatana, E. J. Keogh, A. J. Bagnall, and S. Lonardi. A novel bit level time series representation with implication of similarity search and clustering. In *PAKDD*, pages 771–777, 2005.

[163] E. M. Rosier, M. J. Iadarola, and R. C. Coghill. Reproducibility of pain measurement and pain perception. *Pain*, 98(1):205–216, 2002.

[164] S. R. Sabat. Implicit memory and people with alzheimer's disease: Implication for caregiving. *American Journal of Alzheimer's Disease and Other Dementias*, 21(1):11–14, 2006.

[165] D. P. Salmon. Neuropsychological features of mild cognitive impairment and preclinical alzheimer's disease. In M.-C. Pardon and M. W. Bondi, editors, *Behavioral Neurobiology of Aging*, volume 10 of *Current Topics in Behavioral Neurosciences*, pages 187–212. Springer Berlin Heidelberg, 2012.

[166] A. Schlögl, M. Slater, and G. Pfurtscheller. Presence research and eeg. In *Proceedings of the 5th International Workshop on Presence*, pages 9–11, 2002.

[167] B. Schölkopf, A. J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[168] E. Schulz, L. Tiemann, T. Schuster, J. Gross, and M. Ploner. Neurophysiological coding of traits and states in the perception of pain. *Cerebral Cortex*, 21(10):2408–2414, 2011.

[169] E. Schulz, A. Zherdin, L. Tiemann, C. Plant, and M. Ploner. Decoding an individual's sensitivity to pain from the multivariate analysis of eeg data. *Cerebral Cortex*, 22(5):1118–1123, 2012.

[170] P. Seeman, J. Schwarz, J.-F. Chen, H. Szechtman, M. Perreault, G. S. McKnight, J. C. Roder, R. Quirion, P. Boksa, L. K. Srivastava, K. Yanai, D. Weinshenker, and T. Sumiyoshi. Psychosis pathways converge via d2high dopamine receptors. *Synapse*, 60(4):319–346, 2006.

[171] D. J. Selkoe. Alzheimer's disease is a synaptic failure. *Science*, 298(5594):789–791, 2002.

[172] S. A. A. Shalom, M. Dash, and M. Tue. Efficient k-means clustering using accelerated graphics processors. In *DaWaK*, pages 166–175, 2008.

[173] S. V. Shinkareva, R. A. Mason, V. L. Malave, W. Wang, T. M. Mitchell, and M. A. Just. Using fmri brain activation to identify cognitive states associated with perception of tools and dwellings. *PLoS One*, 3(1):e1394, 2008.

[174] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[175] A. Sims. *Symptoms in the Mind: An Introduction to Descriptive Psychopathology*. W. B. Saunders, 2003.

[176] Y. B. Sirotin and A. Das. Anticipatory haemodynamic signals in sensory cortex not predicted by local neuronal activity. *Nature*, 457(7228):475–479, Jan. 2009.

[177] S. Smith, P. Fox, K. Miller, D. Glahn, P. Fox, C. Mackay, N. Filippini, K. Watkins, R. Toro, A. Laird, et al. Correspondence of the brain's functional architecture during activation and rest. *PNAS*, 106:13040, 2009.

[178] C. Sorg, A. Manoliu, S. Neufang, N. Myers, H. Peters, D. Schwerthöffer, M. Scherr, M. Mühlau, C. Zimmer, A. Drzezga, H. Förstl, J. Bäuml, T. Eichele, A. M. Wohlschläger, and V. Riedl. Increased intrinsic brain activity in the striatum reflects symptom dimensions in schizophrenia. *Schizophr Bull*, 2012.

[179] C. Sorg, V. Riedl, M. Mühlau, V. D. Calhoun, T. Eichele, L. Läer, A. Drzezga, H. Förstl, A. Kurz, C. Zimmer, and A. M. Wohlschläger. Selective changes of resting-state networks in individuals at risk for alzheimer's disease. *PNAS*, 104(47):18760–18765, 2007.

[180] L. E. Souren, E. H. Franssen, and B. Reisberg. Contractures and loss of function in patients with alzheimer's disease. *J Am Geriatr Soc*, 43(6):650–5, 1995.

[181] I. Strigo, A. Simmons, S. Matthews, A. Craig, and M. Paulus. Association of major depressive disorder with altered functional brain response during anticipation and processing of heat pain. *Arch Gen Psychiatry*, 65(11):1275–84, Nov 2008.

[182] A. Sudjianto and G. S. Wasserman. A nonlinear extension of principal component analysis for clustering and spatial differentiation. *Institute of Industrial Engineers, Inc. (IIE)*, v28(n12):p1023(6), 1996.

[183] J. Sun, S. Tong, and G.-Y. Yang. Reorganization of brain networks in aging and age-related diseases. *Aging and Disease*, 3(2):181, 2012.

[184] A. Szalay and J. Gray. 2020 computing: Science in an exponential world. *Nature*, 440:413–414, 2006.

[185] A. Tasora, D. Negrut, and M. Anitescu. Large-scale parallel multi-body dynamics with frictional contact on the graphical processing unit. *Proc. of Inst. Mech. Eng. Journal of Multi-body Dynamics*, 222(4):315–326.

[186] K. R. Thulborn, J. C. Waterton, P. M. Matthews, and G. K. Radda. Oxygenation dependence of the transverse relaxation time of water

protons in whole blood at high field. *Biochimica et Biophysica Acta (BBA) - General Subjects*, 714(2):265 – 270, 1982.

[187] I. Tracey. Can neuroimaging studies identify pain endophenotypes in humans? *Nature Reviews Neurology*, 7(3):173–181, 2011.

[188] I. Tracey and P. W. Mantyh. The cerebral signature for pain perception and its modulation. *Neuron*, 55(3):377–391, 2007.

[189] A. Trouve and Y. Yu. Unsupervised clustering trees by nonlinear principal component analysis. *Pattern Recognition and Image Analysis*, 2:108–112, 2001.

[190] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *NeuroImageVolume 15*, pages 273–289, January 2002.

[191] M. T. van Kesteren, G. Fernández, D. G. Norris, and E. J. Hermans. Persistent schema-dependent hippocampal-neocortical connectivity during memory encoding and postencoding rest in humans. *Proceedings of the National Academy of Sciences*, 107(16):7550–7555, 2010.

[192] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[193] A. Veeraraghavan and A. K. R. Chowdhury. The function space of an activity. In *CVPR (1)*, pages 959–968, 2006.

[194] P. F. Verhaak, J. J. Kerssens, J. Dekker, M. J. Sorbi, and J. M. Bensing. Prevalence of chronic benign pain disorder among adults: a review of the literature. *Pain*, 77(3):231 – 239, 1998.

[195] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1073–1080. ACM, 2009.

[196] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *In Proc. Workshop on Clustering High Dimensionality Data and Its Applications*, pages 23–30, 2003.

[197] E. M. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing & Management*, 22(6):465–476, 1986.

[198] G. Waldemar, B. Dubois, M. Emre, J. Georges, I. G. McKeith, M. Rossor, P. Scheltens, P. Tariska, and B. Winblad. Recommendations for the diagnosis and management of alzheimer's disease and other disorders associated with dementia: Efns guideline. *European Journal of Neurology*, 14(1):e1–e26, 2007.

[199] X. Wang, A. Wirth, and L. Wang. Structure-based statistical features and multivariate time series clustering. In *ICDM*, pages 351–360, 2007.

[200] X. Z. Wang and C. McGreavy. Automatic classification for mining process operational data. *Industrial & Engineering Chemistry Research*, 37(6):2215–2222, 1998.

[201] S. Wichert, K. Fokianos, and K. Strimmer. Identifying periodically expressed transcripts in microarray time series data. *Bioinformatics*, 20(1):5–20, 2004.

[202] E. H. C. Wu and P. L. H. Yu. Independent component analysis for clustering multivariate time series data. In *ADMA*, pages 474–482, 2005.

[203] L. Xiao, H. Begleiter, B. Porjesz, W. Wenyu, and A. Litke. Event related potentials during object recognition tasks. *Brain Research Bulletin*, 38:531–538(8), 1995.

[204] Y. Yamada, E. Suzuki, H. Yokoi, and K. Takabayashi. Decision-tree induction from time-series data based on a standard-example split test. In *ICML*, pages 840–847, 2003.

[205] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, pages 385–394, 2000.

[206] S. Zhong and J. Ghosh. HMMs and coupled HMMs for multi-channel EEG classification. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 2, pages 1254–1159, 2002.