Ordered Means Models for Recognition, Reproduction, and Organization of Interaction Time Series

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
der Technischen Fakultät der Universität Bielefeld

vorgelegt von Ulf Großekathöfer am 16. April 2013

To my parents

Acknowledgements

First, I would like to thank my supervisor Thomas Hermann who has been a fantastic mentor. Without his active support, his encouraging criticism, and the freedom he provided, this thesis would not have been possible. Another big thank you goes to Stefan Kopp who has been an exceptional advisor during my PhD. His straight-forward spirit helped me to focus on the essentials of a PhD thesis. Working in both their groups has been not only inspiring but also a great pleasure. I am also very grateful to Shlomo Geva for being a incredible external advisor and for the great support during my research stay at the Queensland University of Technology, Australia. I would further like to thank Barbara Hammer for being an examiner in my thesis committee and for the time and effort associated with this commitment. Another big thank you goes to my good friends and colleagues Peter Meinicke and Thomas Lingner for our lively discussions, help in all machine learning related questions, and long and surprisingly productive pub-sciences sessions.

To my colleagues in the Ambient Intelligence Group including Till Bovermann, Angelika Dierker, Florian Grond, Tobias Großhauser, Sebastian Hammerl, Jan Hammerschmidt, Christian Leichsenring, Alexander Neumann, Christian Peters, Eckard Riedenklau, René Tünnermann, and Sebastian Zehe, deep thanks for an inspiring and fun workplace. Many thanks go to the people from the Sociable Agents Group including Kirsten Bergmann, Elmar Bienek, Hendrik Buschmeier, Farina Freigang, Sebastian Kahl, Dagmar Philipp, Sebastian Ptock, Amir Sadeghipour, and Ramin Yaghoubzadeh for providing such a pleasurable environment. A big thank you goes to Nils-Christian Wöhler; his incredible devotion to projects and his ability to solve problems on his own, enabled me to focus on my thesis. My thanks also go to the people of CITEC, in particular to Claudia Muhl who managed the Graduate School with a perfect balance between official requirements and scientific freedom. Many thanks also to all of you I did not mention personally, who have provided feedback, comments, and ideas to this work.

A big, big thank you goes to my parents for their endless support, incredible kindness, and constant belief in me.

Finally, I am deeply grateful to my truelove Mona who has been on my side during all these years. She encouraged and supported me, and has been a true friend. Thank you for all of that — and for simply being AWESOME.

During the time of my PhD at Bielefeld University I was funded by a Scholarship from the Cognitive Interaction Technology – Center of Excellence.

Contents

1. Introduction

When interacting with one another, humans rely on a variety of expressive behaviors including utterances, gestures, facial expressions, gaze, etc. Being able to interact with others, thus, requires to perceive, recognize, and interpret such behaviors, as well as to generate and employ them purposefully for expressing meaning and communicative intentions. The main challenge arises from the dynamic nature of interactive behavior: conventionalized behaviors like utterances or symbolic gestures (e.g., a thumb-up gesture) work by agreement on their meaning between individuals within a certain community, e.g., all speakers of English. However, these behaviors are still subject to considerable variation due to situational constraints, personal style, cultural traits, etc., and beyond this variability, many behaviors themselves are not conventionalized at all. Such behaviors are created by one interlocutor on the spot, get dynamically established between the interactants, and becoming part of their so-called common ground. In summary, interacting with one another means engaging in a highly dynamic process in which the forms as well as the meanings of expressive behaviors are flexibly and spontaneously observed, recognized, picked up, adapted, and adopted. Computer systems that are to participate in such scenarios are confronted with various technical challenges that arise from the complex and dynamical nature of human interaction.

Machine learning offers a wide variety of approaches for interaction technologies. With machine learning techniques, computer systems can identify relevant data, recognize important data changes, or focus their resources and capabilities toward a given target. In interaction environments, most of the collected data, e.g., gestural communication, attention signals, speech interaction, multivariate sensor data streams from a smart room's system, etc., can be described as time series data; reliable, robust, and flexible machine learning algorithms for time series are crucial for a number of desirable ways of interactions:

- **Recognition** of human behaviors such as speech, gestures, or handwriting,
- **Reproduction** of these observations by means of available actuators, and
- **Organization** of observations according to an inherent data structure.

However, as stated above, interaction with humans is a highly dynamic process and enabling a computer system to participate in such a scenario is a difficult task. The system is confronted with various technical challenges: (i) In order to enable a continuous interaction without significant delay, fast response times from the data processing systems are crucial. (ii) Interactive systems are required to make sense of their observations incrementally to foresee evolving behaviors. (iii) In order to be successful, even in unknown and unforeseen situations, a system should be able to adapt to particular users or environments. (iv) The approach has to provide robustness towards incomplete observations, i.e., it should be able to recognize behaviors even if they are incompletely executed. (v) An ideal algorithmic solution integrates all of the above-mentioned abilities — recognition, reproduction, and organization — in one model.

Common choices for machine learning techniques that address the particular properties of multivariate time series, i.e., variable length and non-linear distortions of the time axis, are Dynamic Time Warping [DTW, Chiba and Sakoe, 1978] and Hidden Markov Models [HMMs, Rabiner, 1989]. Numerous applications in interaction scenarios, e.g., in speech [Rabiner, 1989], handwriting [Plötz and Fink, 2009], or gesture recognition [Kellokumpu et al., 2005], underline the importance and significance of those methods. State-of-the-art performance can be achieved if Support Vector Machines [Vapnik, 2000] are combined with specialized time series kernels for interaction data [e.g Bahlmann et al., 2002]. However, the computational costs of recognition with such approaches are usually high due to the evaluation of a possibly large number of kernel functions.

The first research objective of this thesis is to develop algorithms for machine learning of interaction time series data that allows recognition, reproduction, and organization of observed behavior. Thereby, the algorithms should meet the particular requirements outlined above that allow successful application in interaction scenarios. A second research objective of this thesis is to empirically evaluate the algorithms with time series data from various interaction scenarios. These data sets include gestures of different body parts (hands, arms, heads), computer mouse gestures, data collected from musicians handling their instruments, speech, handwriting, but also — to underline the general applicability of the proposed algorithms to time series — time series benchmark data sets from various alternative domains. We will evaluate the proposed algorithms in terms of research questions related to accuracy, response times, robustness, adaptability, incremental evaluation, prototyping, influence of hyperparameter and the training process, as well as their organization abilities.

The thesis is structured as follows: in Chapter 2 we will introduce this thesis' aim in detail and define the research objectives and research questions that we will address. Chapter 3 summarizes the theoretical background of this thesis. Here, we present relevant machine learning techniques for time series data, and give an overview about related work where techniques for learning and representing iteration data are used. In Chapter 4 we introduce Ordered Means Models (OMMs), a probabilistic state-space model that we use as a starting point for this thesis' algorithmic work. Subsequently, we specify the experimental setups, the algorithms that we chose for comparison, and give an overview of the data sets in Chapter 5. Chapters 6, 7, and 8 are related to experimental evaluation of OMM's recognition, reproduction, and organization capabilities, respectively. Last, Chapter 9 concludes the thesis with a summary and a discussion of its results. The last section of this chapter outlines the scope for future research directions.

The presented work resulted in the following peer-reviewed publications and contributions:

- N.-C. Wöhler, U. Großekathöfer, A. Dierker, M. Hanheide, S. Kopp, T. Hermann. A Calibration-free Head Gesture Recognition System with Online Capability. In *International Conference on Pattern Recognition*, Istanbul, Turkey, 2010.
- T. Grosshauser, U. Großekathöfer, T. Hermann. New Sensors and Pattern Recognition Techniques for String Instruments. In *NIME 2010: Proceedings of the 2010 Conference on New Interfaces for Musical Expression*, 2010.
- U. Großekathöfer, A. Barchunova, R. Haschke, T. Hermann, M. Franzius, H. Ritter. Learning of Object Manipulation Operations from Continuous Multimodal Input. In *Proceedings of the IEEE/RAS International Conference on Humanoid Robots 2011.*

- U. Großekathöfer, A. Sadeghipour, T. Lingner, P. Meinicke, T. Hermann, S. Kopp. Low Latency Recognition and Reproduction of Natural Gesture Trajectories. In *ICPRAM 2012: Proceedings of the International Conference on Pattern Recognition Applications and Methods 2012).*
- U. Großekathöfer, N.-C. Wöhler, T. Hermann, S. Kopp. On-the-fly Behavior Coordination for Interactive Virtual Agents: A Model for Learning, Recognizing and Reproducing Hand-Arm Gestures Online. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems 2012.*
- U. Großekathöfer, S. Geva, T. Hermann, S. Kopp. Learning Hierarchical Prototypes of Motion Time Series for Interactive Systems. In *ECAI2013: Proceedings of the Workshop on Machine Learning for Interactive Systems*, 2012.

2. Recognition, Reproduction, and Organization of Interaction Time Series

Consider a scenario like companion-based interaction with a smart room where the room's intelligence is managed by means of a virtual agent. In such a setting, we find a dense net of sensors that collects data about the room, e.g., the room's state might be captured by temperature and air pressure sensors, and motions in the room might be detected by means of cameras. The virtual agent has to make sense of the information captured by the sensor network, he has to be able to recognize activities and behavior patterns of the room's inhabitants, organize the observation data, adapt to common scenarios or persons that use the room regularly — and all in a robust and reliable way.

In order to operate computer systems, appropriate interfacing is necessary. Information and data has to be transferred to the computer, instructions how to process the data need to be communicated, and, eventually, the computer has to be able to present the outcome of a computing process back to the user. Computers are tools that are able to quickly solve computational tasks. By design, computers sequentially process simple arithmetic commands that can be combined to more complex tasks and applications. By means of abstraction, mathematical modeling, and higher-order programming techniques, computers contribute to various aspects of today's life: database systems store and organize large data sets for banks or libraries, high-performance compute clusters analyze large amounts of biomedical data, and digital controllers navigate space ships. Computers are also applied to many everyday objects such as smart phones, elevators, streetlights, televisions — and even some refrigerators are equipped with internet connectivity. However, in order to use this omnipresent computing power, the design of the interaction between user and computer system is becoming increasingly important. How the loop of data and instructions input, data processing, and presentation of results is modeled, has a major impact on what can be achieved by the user and of what quality the interaction is.

In the early days of modern computing, there was only minimal interaction between human users and computers. Data and programs were communicated to the machine by means of punched cards — paper cards with holes that represent information in bits. Today, the usage of computers is dominated by graphical user interfaces (GUIs). GUIs are usually displayed on monitors, and are controlled by means of keyboards, which are used to enter data and commands to the computer, and computer mice that act as simple pointing devices [Myers, 1998]. Even though GUIs are still widely used today, they do not present an optimal interfacing for many applications.

A research field that is related to studying and improving the interaction between computer systems and their users is Human-Computer-Interaction [HCI, Dix, 2004]. HCI is associated to the design, implementation, and evaluation of interactive computing systems for use of humans. It is connected to many academic disciplines: research from psychology, computer science, design, art, or cognitive science contributes to novel and more intuitive HCI experiences. Moreover, applications in tangible computing [Ullmer and Ishii, 2000], social robotics [Breazeal, 2003], or auditory displays [Kramer, 1994] prove that HCI research improves and re-defines the way we use computer systems today. A major research objective in HCI is to enable more human-like interaction with the aim that computer systems should provide communicative functionality that is natural to human users.

A wide variety of human expressions towards a computer system can be considered as interaction data. These modalities include classical keyboard and mouse interaction, interaction by spoken or written words, gestural interaction and body motions, interaction via touch-sensitive surfaces, as well as brain computer interfaces that use electroencephalography readings for communication purposes. E.g., gestures are widely-used in human communication, where they structure and facilitate communication by (i) either pragmatic functions such as emphasizing a speaker's linguistic course, (ii) semantic functions, such as classical pointing gestures to indicate spatial location, (iii) or featuring semantic content, for example, the Victory 'V' or the OK gestures — let alone sign languages for hearing impaired [McNeill, 1992]. Enabling interactive systems to employ gestures — for either recognition or reproduction — expands the potential interaction experiences. A just as widely used modality in human-computer interaction is speech. Speech is a vocalized form of human interaction, where language is articulated as sound that is decoded by a receiver [Campbell, 1982]. Speech is ubiquitous in human interaction, and since it is part of the human nature for tens of thousands of years, human interactants are expert speech users. In order to allow computer systems to benefit from this expertise the computer has to be able to understand and generate speech. Such a task comprises a complex combination of low- and high-level speech and language processing. Thus, in order to understand what a spoken utterance means, the computer is required to derive meaning and intention from an audio signal. This complex process might not only involve audio processing and linguistic knowledge, but also domain knowledge and a user model. Last yet importantly, the ability to perceive written text is a remarkable human accomplishment, and written words are a widely used communication medium. Allowing computer systems to recognize handwritten (or machine-printed) text is a major aim of pattern recognition research that for limited domains such as address-reading in machines that sort mail, seems to be solved. In more complex domains such as longer domain-independent text, current technologies reach their limits.

Today's advances in HCI are ushered by developments of novel and alternative input and output devices, as well as improved hardware in general. Over the past years, we witnessed a dramatic evolution in computing technologies. These include increased computational power, e.g., the availability of multi-core CPUs and enlarged memory that leads to faster systems that can process more data and, therefore, solve tasks that are more complex. Advances in sensor technology such as cheap 9-DOF modules to measure circular and linear acceleration as well as low-field magnetic sensing, enable cheap motion-based applications. In combination with reduced power requirements and minimization trends in hardware design, this leads to portable HCI solutions. This is best illustrated by smartphones that comprise, among a global positioning system that provides location information anywhere on earth, acceleration sensors and high computational power. In addition, various novel input devices such as multi-touch displays, touch-sensitive surfaces, eye tracking devices, depth-of-field cameras, tactile sensor grids, etc. allow alternative data input. Broadband network access is widely available and enables high-throughput applications and distributed computing based on cloud computing. Due to wireless networks, these can be accessed

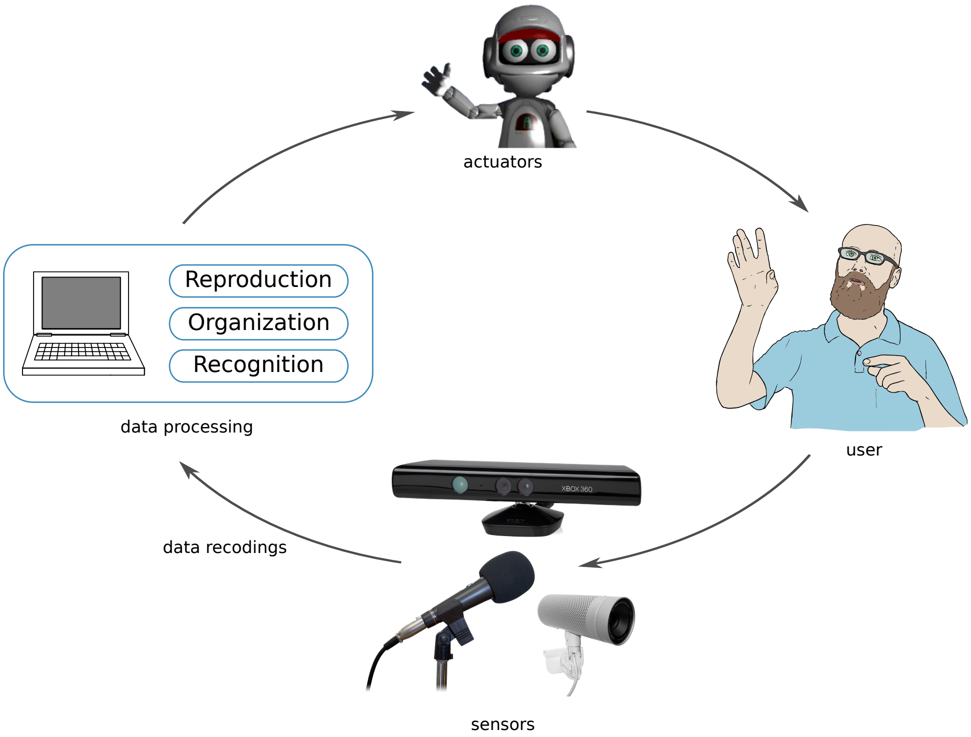


Figure 2.1.: This figure illustrates an interaction loop: a user acts, the system's sensors captures relevant data, and presents the results from a data processing back to the user.

almost everywhere — yielding mobile and everywhere usable applications. Also improved displays, e.g., retina displays, E-Ink technology, 3D displays, as well as tangible interfaces are available and promise exciting novel research opportunities and applications.

In lights of such advances, it is clear that, in addition to powerful hardware components, appropriate algorithms for processing the recorded data are crucial. If we consider the before-mentioned scenario of a companion-based interaction with a smart room, we can identify different requirements for successful interaction. On one hand, the system is confronted with observations from various sensor sources. In order to make sense of the collected data, the virtual agent's computing system should be able to recognize recurring behavior patterns from human users in order to map raw signal recordings to (meaningful) behavior representations. This might include recognition of gestures and speech, or analysis of facial expressions to, e.g., estimate a user's emotional state. On the other hand, a user's interaction experience with the virtual companion would greatly benefit if the agent would be able to align to the user by reproducing her or his behaviors. E.g., if users perform particular greeting gestures, they would endure familiarity through an agent who is able to imitate their greeting habits. Last yet importantly, behavior patterns from humans might appear with great variance. E.g., handwriting varies from writer to writer — even within a single writer —, and the ability to organize these observations unsupervised would allow the companion to autonomously discover otherwise hidden similarities. Please refer to Figure 2.1 for an illustration of the interaction scenario.

	unsupervised	**supervised**
signal → symbol	Organization	Recognition
symbol → signal	Reproduction	

Figure 2.2.: This matrix illustrates the classification of the terms recognition, organization, and reproduction along the dimensions supervised vs. unsupervised and signal→symbol vs. symbol→signal.

These requirements also correspond to translations between signals and meanings according to a sub-symbolic/symbolic paradigm. In such a paradigm, a sub-symbolic representation is related to a raw signal or feature representation of an observation, and a symbolic representation equals a semantic and meaningful denotation. Given this, pattern recognition algorithms might be considered as a supervised translation of a sub-symbolic signal to a symbol or category. This translation is supervised because pattern recognition algorithms usually require labeled data, i.e., observations whose categories are known, to learn a generalized recognition scheme. Analogously, organization corresponds to an unsupervised conversion from signal to categories. However, in such a scenario, a system should discover categories autonomously and unsupervised without prior knowledge in terms of predetermined labels. In contrast, reproduction describes the process of generating a sub-symbolic signal realization for a given symbol — regardless of how the symbolic representation was gathered. Figure 2.2 illustrates the structure along the dimensions supervised vs. unsupervised and signal→symbol vs. symbol→signal translation.

Thus, depending on the aspired application, recognition, reproduction, and organization of observation are fundamental prerequisites for a successful interactive system:

- **Recognition:** The system has to be able to recognize human behavior patterns such as utterances, communicative gestures, or hand writing.
- **Reproduction:** In addition, the interactive system should be able to reproduce observed behavior patterns by means of its own actuators. E.g., a virtual agent can use his internal behavior model that was learned by observations to reproduce behavior himself.
- **Organization:** Ideally, the interactive system is able to organize its observations according to an inherent data structure. E.g., if the human user performs a behavior pattern from one category in very different ways, the system should be able to structure the performances accordingly.

Based on these requirements, a technical application of such algorithms in interaction scenarios are confronted with various challenges:

- **Low Latency Response:** In order to enable a continuous interaction without significant delay, fast response times from the data processing systems are crucial. E.g., an interactive system needs to respond in a certain time frame to an utterance of a human user to act plausible and be believable.
- **Incremental Update:** Human interactants are able to make sense of their observations incrementally and can foresee, e.g., speech utterances or gestural behaviors. Interactive systems require a similar incremental processing scheme that updates a potential outcome or result while observing.
- **Adaptive Model:** A system should be able to adapt to a particular user or environment to be successful, even in unknown and unforeseen situations, or in case of unpredicted events.
- **Robustness:** An interactive system should provide robustness toward incomplete observations, i.e., it should be able to recognize a user's behavior even if the behavior is imprecisely or incompletely performed.
- **Integrated Approach:** Ideally, an algorithmic solution integrates all of the above-mentioned abilities — recognition, reproduction, and organization — in one model. Thus, instead of using separated modules for each sub-task one model fits it all.

Usually, interaction happens in a temporal, sequential manner: gestural communication, attention signals, speech interaction, or multivariate sensor data streams from a smart room's sensor network occur in time. Subsequent to a necessary multi-sensor data fusion step [Xiong and Svensson, 2002] in which readings from various sensor sources are joined, the observed interaction data can be naturally modeled as multivariate time series. For this reason, we will represent interaction data as time series data and assume a prior fusion in this thesis. Machine learning offers a wide variety of algorithms to process time series data. Algorithms are able to recognize, reproduce, and organize interaction data according to the above illustrated requirements, and numerous applications in speech recognition [Rabiner, 1989], handwriting recognition [Plötz and Fink, 2009], and gesture recognition [Kellokumpu et al., 2005] underline the importance of accurate and fast methods. In particular, specialized techniques are required to cope with the variable length and with non-linear compressions or expansions of the time axis. Among others, Dynamic Time Warping [DTW, Chiba and Sakoe, 1978] and Hidden Markov Models [HMMs, Rabiner, 1989] are common approaches to deal with the variability of time series data. The ability of HMMs to model time series data typically arises from a combination of discrete model states with state-specific distributions of observable variables and the transitions between these states. Although HMMs provide a high flexibility and provide recognition and reproduction capabilities, choosing the optimal model architecture for a particular task usually requires expert knowledge. Suboptimal HMM structures may easily result in a decrease of speed and accuracy. In particular, unreasonably low or high transition probabilities estimated during the model training process may result in a lack of model robustness. For that reason more simple techniques such as DTW are still widely used [e.g., Rath and Manmatha, 2003]. Focusing on accuracy, discriminative machine learning approaches have been shown to outperform HMMs in many recognition problems. State-of-the-art performance can be achieved if Support Vector Machines [Vapnik, 2000] are combined with specialized time series kernels [Bahlmann et al., 2002]. However, the computational cost of recognition is usually high due to the evaluation of a possibly large number of kernel functions.

2.1. Research Objectives and Research Questions

The work presented in this thesis aims to systematically address the above described requirements and challenges. We will contribute work according to the following technique and application driven research objectives:

As a technical motivated research objective, this work aims to develop algorithms for machine learning of interaction data that are represented as time series. A major challenge is to present a solution that integrates different usage scenarios and that can be used for recognition, reproduction, and organization of interaction data. In addition, the presented algorithms should meet the particular requirements according to robustness, incremental evaluation, low latency response times, and adaptability.

In addition to the algorithmic work, we will empirically evaluate and apply the algorithms in various interaction scenario. We will use them with various time series data sets that are related to interaction data. These include gestures that are performed with various body parts (hands, arms, heads), computer mouse gestures, data collected from interaction with musical instruments, speech, handwriting, but also — to underline the general applicability of the proposed algorithms to time series — time series benchmark data sets from various other domains.

As the fundamental basis for our algorithmic work, we will use Ordered Means Models — an easy-to-use machine learning approach for time series data that has proven to perform very well in different usage scenarios [Großekathöfer and Lingner, 2005].

In order to evaluate if Ordered Means Models are an appropriate approach to machine learning of interaction data that is able to meet the above-denoted requirements, these requirements were divided into a set of research questions:

- **(Q1) Accuracy:** How do OMM-based classifiers compare to standard methods for classification of time series data in terms of accuracy? (→ Recognition, Chapter 6)
- **(Q2) Response Times:** How do OMM-based classifiers compare to standard methods for classification of time series data in terms of response times? (→ Recognition, Chapter 6)
- **(Q3) Robustness Property:** How do OMM-based classifiers perform in terms of incomplete observations in comparison to standard classifiers? (→ Recognition, Chapter 6)
- **(Q4) Adaptability:** To what degree can OMMs dynamically adapt to changing behaviors? (→ Recognition, Chapter 6)
- **(Q5) Incremental Evaluation:** Does an incremental evaluation of OMMs allow an on-the-fly classification of observed behavior, and can this improve the time required for recognition? (→ Recognition, Chapter 6)
- **(Q6) Prototype Property:** Do OMMs provide interpretable prototypes of interaction data, i.e., to what degree is the learned array of reference vectors a reliable prototype representation for a given data set? (→ Reproduction, Chapter 7)
- **(Q7) Influence of Hyperparameters:** What influence does the choice of the OMM hyperparameters, namely, the deviation parameter σ and the parameter that indicates the number of model states K have? (→ Reproduction, Chapter 7)

- **(Q8) Training Process:** To what degree is the development of the learned OMM representations related to the development of the objective function? (→ Reproduction, Chapter 7)
- **(Q9) Organization:** How can OMMs be used to organize observations that are represented as time series in an unsupervised manner, without intervention of an external human observer? (→ Organization, Chapter 8)
- **(Q10) Hierarchical Organization:** How can we enable OMMs-based methods to discover inherent but hidden hierarchical structures in observed interaction behaviors, again unsupervised? (→ Organization, Chapter 8)

2.2. Document Structure

This thesis is structured as follows:

Chapter 3: Background: Machine Learning of Time Series covers the theoretical background of machine learning of time series data with a major focus on interaction data. The presented algorithms are limited to learning approaches that will be used in experiments later in this thesis. In detail these are: Nearest Neighbor Classifiers with Dynamic Time Warping Distance Functions, Support Vector Machines with particular Time Series Kernels, Hidden Markov Models and extension, Ordered Means Models, $\mathcal{K}$-means and $\mathcal{K}$-tree clustering, alongside essential fundamentals. In addition, we will introduce evaluation measures for different machine learning tasks.

Chapter 4: Theory and Definition of Ordered Means Models specifies Ordered Means Models in detail. After describing the model design in detail, we will present the process of estimating appropriate model parameter by means of a set of observation, followed by considerations according to numerical aspects in practical applications. Last, we will discuss the relation of OMMs and HMMs in Section 4.4.

Chapter 5: Experiments: Methods, Data Sets, and Application introduces the evaluation schemes we chose in experiments. We will present the standard machine learning algorithms that we chose to evaluate and compare OMM-based recognition, reproduction, and organization techniques. Further, we will outline the application of the algorithms in experiments with real-world data, e.g., how the data was normalized, and how a particular model configuration for a given task was selected. Additionally, the 28 data set that we used for evaluation are presented together in a short condensed fashion alongside their basic properties.

Chapter 6: Recognition of Interaction Time Series Data with OMMs is related to investigate how well Ordered Means Models are able to recognize time series data with a focus on interaction time series. After presenting the theoretical outline of this chapter, we will describe a maximum-likelihood approach to classification with OMMs, followed by an extension to incremental classification of time series that allows recognition of behavior patterns while they are observed. We will close with a description of an adaptive online learning scheme. In an experimental evaluation, we will investigate if OMM-based classifiers are suited for recognition of interaction time series. We will focus OMMs' robustness probabilities, and we will introduce two applications in interaction scenarios: (i) Recognition of communicative head gestures and (ii) recognition of bowing-types and malpositions in bowing instrument playing. In addition, we will evaluate the incremental classification approach as well as the adaptive training scheme in an interaction scenario where a human plays a variant of the well-known Rock-Paper-Scissors game against the virtual agent VINCE.

Chapter 7: Reproduction of Interaction Time Series Data with OMMs evaluates the capabilities of OMMs for reproduction of interaction data. We will investigate OMM's prototype property as well as the influence of the hyperparameter selection and the training process on the emerging prototypes. In experiments, we will analyze a mouse gesture data set that comprises 33 computer mouse gestures in the classes *spiral*, *figure eight*, and *circle*. We will also apply OMMs to reproduce and recognize natural gesture trajectories represented as 3-dimensional location coordinates of the right hand wrist.

Chapter 8: Organization of Interaction Time Series Data with OMMs refers to the question, if OMMs are able to organize interaction time series data in an unsupervised manner. Therefore, we will introduce and evaluate two approaches to unsupervised time series processing based on OMMs: $\mathcal{K}$-OMMs provides a clustering solution for time series that is similar to classical $\mathcal{K}$-means clustering. $\mathcal{K}$-OMM-trees, is a tree-based approach to unsupervised hierarchical organization of time series data. We will empirically evaluate $\mathcal{K}$-OMMs by means of 20 benchmark data sets. To evaluate the clustering performance of $\mathcal{K}$-OMMs algorithm, we will discuss different initialization strategies for $\mathcal{K}$-OMMs clustering. Subsequently, we will assess $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-trees clustering approaches by means of interaction time series data sets from the UCI machine learning repository. Last, we will apply $\mathcal{K}$-OMM-trees to a recognition task and investigate if their hierarchical organization capabilities lead to improved generalization in gesture recognition.

Chapter 9: Conclusion concludes this thesis with a summery, and a discussion of its results, while Section 9.3 outlines future research direction based on the outcome of this thesis.

3. Background: Machine Learning of Time Series

Time series are ubiquitous in interaction scenarios and most interaction data can naturally be described as time series. E.g., readings from a smart room's sensor network are sampled in time, gestures can be easily represented as location coordinates that change in time, or speech might be modeled as audio features that, also, vary in time. As stated in Chapter 2, machine learning algorithms that can process time series data hold a key-role in successful application design for interactive systems.

The following chapter will cover theoretical background about machine learning in general, and machine learning of time series in particular. The presented algorithms are limited to learning approaches that will be used in experiments (Chapter 6 ff). In detail these are: Nearest Neighbor classifier (Section 3.3.2) with Dynamic Time Warping distance functions (Section 3.2.1), Support Vector Machines with particular time series kernels (Section 3.3.3), Hidden Markov Models and extension (Section 3.2.2), $\mathcal{K}$-means (Section 3.4.1) and $\mathcal{K}$-tree clustering (Section 3.4.2), alongside essential fundamentals. In addition, we will investigate how algorithms for machine learning of time series are used in interaction scenarios in Section 3.5. Last, we will present performance measures (Section 3.6) for classification and clustering that we will use to evaluate our methods in comparison to other algorithms.

3.1. Machine Learning

In general, the term *learning* describes the ability of a system to improve in managing a task by example or observation [cf., e.g., Nilsson, 1996, Langley et al., 1996]. According to Cherkassky and Mulier [1998],

> *a learning method is an algorithm (usually implemented in software) that estimates an unknown mapping (dependency) between a system's inputs and outputs from available data, namely from known (input, output) samples.*

The outcome of a machine learning algorithm is usually expressed as a function $f(\mathbf{x}) = \mathbf{y}$ where $f(\cdot)$ differs according to the given task. Most algorithms include a training phase or learning phase in which the function $f(\cdot)$ is adapted to fit or represent a data set of observations $X = \{\mathbf{x}^1, \dots, \mathbf{x}^N\}$, the so-called training data set. The goal of the training phase is to realize a function $f(\cdot)$ that is able to process unknown observations $\mathbf{x}$ successfully in terms of the given task — a property is usually described by the term generalization. A common distinction of machine learning approaches is supervised and unsupervised learning. Algorithms, in which the training observations $\mathbf{x}^i$ are presented together with their designated outcome $\mathbf{y}^i$ as tuples $(\mathbf{x}^i, \mathbf{y}^i)$, are called supervised learning. These tasks include, e.g., classification and regression problems. If $\mathbf{y}^i$ is unknown or does not exist

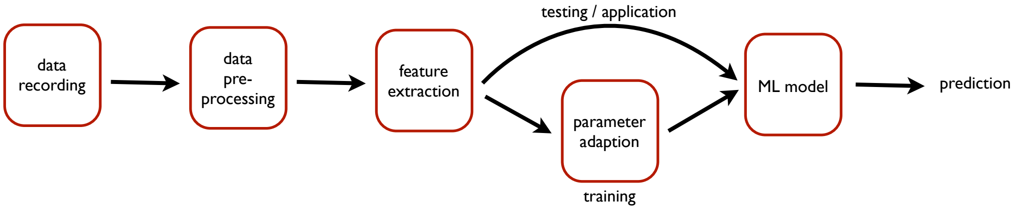


Figure 3.1.: This figure shows schematic steps that are necessary for machine learning.

and only $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ are presented to the learner, the algorithms are called unsupervised learning algorithms. Here, clustering, i.e., the discovery of data subsets that share a common quality is a well-known task. In addition, methods such as Principal Component Analysis, Independent Component Analysis etc. process data independently from a label and, thus, are unsupervised.

Machine learning approaches differ in how the function $f(\cdot)$ and the observations are represented. Figure 3.1 illustrates a typical machine learning approach, starting with data recording, data preprocessing and feature extraction, and finishing with a prediction from a learned model. Common machine learning tasks include

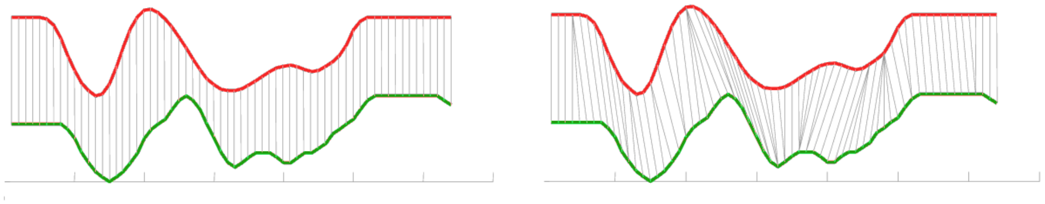
- **Density Estimation:** estimation of an unknown, underlying probability density function for a set of observations,
- **Classification:** assignment of an unseen observation to a given category,
- **Regression:** learning a dependency between two or more random variables by means of observations,
- **Clustering:** discovering of data partitions or clusters in a given set of observations according to a similarity or distance criterion.

An important matter for all machine learning tasks is the choice of an appropriate data representation, i.e., how the observations are represented in a learning system.

Usually, this feature extraction step (cf. Figure 3.1) embraces data recording, digitization and quantization, as well as data pre-processing, e.g., segmentation and normalization of the recorded data. In machine learning, observations are usually represented as feature vectors $\mathbf{x} \in \mathbb{R}^d$ where each dimension $d' \in \{1, \ldots, d\}$ is associated with one particular feature.

3.2. Alignment Methods for Time Series Data

In order to apply machine learning algorithms to time series data, specific preparations and methods are necessary. Time series data might involve different lengths and varying developments in time, i.e., compressions and expansions of the time-axis. Let's consider $O^i = \mathbf{o}^i_1, \ldots, \mathbf{o}^i_{T_i}$ and $O^j = \mathbf{o}^j_1, \ldots, \mathbf{o}^j_{T_j}$ to be two time series that are elements of time series space $\mathbb{O}$ with $\mathbf{o}^i \in \mathbb{R}^d$. Due to expansions or compressions of the time axis, it is unclear how to optimally represent such observations in an uniformed feature space, thus, requiring costly data pre-processing. A particular successful family of algorithms that circumvent a classical feature space representation of time series apply alignments to time series, i.e.,



(a) Euclidean distance sample assignments

(b) Dynamic Time Warping sample assignments

Figure 3.2.: This figure illustrates the differences between the sample assignments of two time series according to Euclidean distance (a) and Dynamic Time Warping (b) [based on an illustration from Keogh and Pazzani, 1999].

two time series are aligned towards each other to obtain a value that is associated with a defined, joined quality. In other approaches, e.g., HMMs, the time series are aligned to data models.

There are various approaches to alignment of time series and sequential data. In case of discrete data, i.e., sequences that only contain elements from a finite alphabet such as protein sequences or words, string metrics are often useful. Such approaches align one observation with another according to a similarity measure and penalize insertions, deletions, or replacements of single or multiple elements. Famous string metrics are edit distances such as Levensthein Distance [Levenshtein, 1966] that penalize insertions, deletions, and mismatches by the same penalty. In alignment approaches that origin in bio-informatics, e.g., the Smith-Waterman algorithm [Smith and Waterman, 1981], insertions, deletions and mismatches are penalized according to biological motivated costs, e.g., based on the likelihood that a particular mismatch occurs.

For continuous time series, various distance measures and functions exist. In addition to simple approaches based on L-norms such as Manhattan, Euclidean, or maximum distances, that require time series of the same lengths, there are many alignment measures based on the concept of edit distance. Here, the most prominent example is Dynamic Time Warping [DTW, Sakoe and Chiba, 1978] that allows expansions or compressions of the time axis. In order to speed up DTW a constrained warping window size reduces the computational demands and can also improve the accuracy. Alternative approaches are Longest Common Sub-sequence [Vlachos et al., 2002], Edit Distance on Real Sequences [Chen et al., 2005], and Edit Distance with Real Penalty [Chen and Ng, 2004]. These methods try to improve DTW by also allowing insertions and deletions in the warping path. For a comprehensive comparison of similarity measures of time series data and their performance in classification, please see Ding et al. [2008].

3.2.1. Dynamic Time Warping

In order to compute a distance function for two time series of different lengths and varying development in time, Dynamic Time Warping (DTW) aligns the two time series according to an order preserving warping path. The computed DTW value can then be used, e.g., as a distance function value in Nearest Neighbor classification (cf. Section 3.3.2).

For (multivariate) time series, the sum of Euclidean distances for all samples is an insufficient distance measure, because (i) the length of the time series may differ and, therefore, a data pre-processing such as truncation of the longer time series, or interpolation of one of both time series to equal lengths [Keogh and Ratanamahatana, 2004] may be required. In addition, (ii) such an approach is prone to small variations in time.

The DTW algorithm minimizes a global distance by means of a warping path that assigns samples from one time series $O^1 = \mathbf{o}^1_1, \ldots, \mathbf{o}^1_{T_1}$ to samples of a second time series $O^2 = \mathbf{o}^2_1, \ldots, \mathbf{o}^2_{T_2}$. A warping path $\mathbf{q} = q_1, \ldots, q_T$ between these time series is an assignment function that can be represented as a $T_1 \times T_2$ matrix

$$\Phi : \{1, \ldots, T\} \rightarrow \{1, \ldots, T_1\} \times \{1, \ldots, T_2\}, \tag{3.1}$$

with T being the length of the warping path that is constrained by $\max(T_1, T_2) \leq T \leq T_1 + T_2 - 1$. Here, $q_t = (i, j)_t$ denotes indexes from a $T_1 \times T_2$ matrix that contains pairwise distances $d(\mathbf{o}^1_i, \mathbf{o}^2_j)$ between the time series' samples. This distance is usually chosen to be Euclidean.

The warping path has to fulfill the following constraints:

- **Monotonicity:** The warping path has to be order preserving in time: if $q_{t-1} = (i, j)$ and $q_t = (i', j')$ then $i' - i \geq 0$ and $j' - j \geq 0$ is necessary,
- **Continuity:** For $q_{t-1} = (i, j)$, and $q_t = (i', j')$, $i' - i \leq 1$ and $j' - j \leq 1$ is true,
- **Boundary Condition:** $q_1 = (1, 1)$, and $q_T = (T_1, T_2)$.

The number of possible paths increases exponentially with the lengths of the time series, thus, it is impossible to explicitly compute the warping costs for all paths. However, we are only interested in the path that minimizes the warping costs

$$d_{dtw}(O^1, O^2) = \frac{1}{T} \sum_{t=1}^{T} w_t,$$

where w_t denotes the pairwise distance $d(\mathbf{o}^1_i, \mathbf{o}^2_j)$ associated with q_t. This value and the corresponding optimal DTW path can be efficiently computed by a recursive dynamic programming solution through

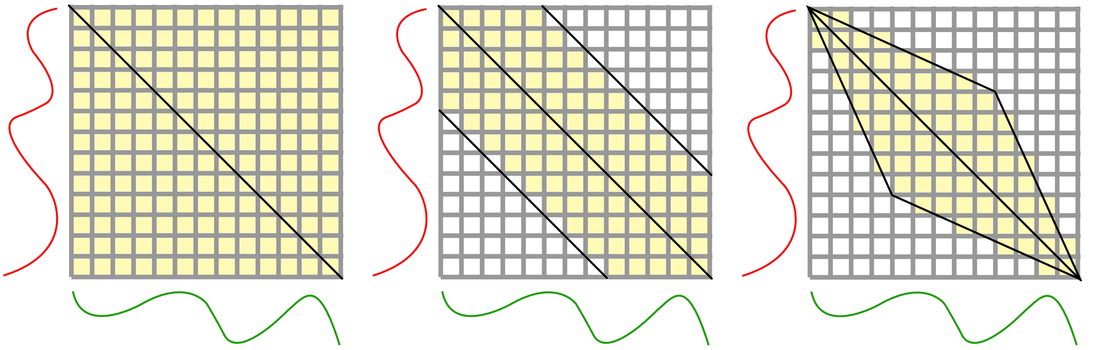
$$\gamma(i, j) = \begin{cases} d(\mathbf{o}^1_1, \mathbf{o}^2_1), & \text{if } i = j = 1, \\ d(\mathbf{o}^1_i, \mathbf{o}^2_j) + \min\left[\gamma(i-1, j-1), \gamma(i, j-1), \gamma(i-1, j)\right] & \text{if } i > 1 \text{ or } j > 1, \\ \infty & \text{otherwise.} \end{cases}$$

$\gamma(i, j)$ accumulates the DTW value up to the i-th and j-th time series samples. In order to retain the optimal path, the assignments (i, j) have to be saved in the assignment function Φ.

The algorithmic complexity of the DTW algorithm is $O(T_1 \cdot T_2)$. To further reduce the computational demands of DTW, the set of possible alignment paths can be limited to paths that are close to the diagonal. In addition, lower bounding techniques allow faster similarity search in huge time series data sets.

Pruning for Dynamic Time Warping

An intuitive alignment path is unlikely to drift very far from the diagonal. Thus, an obvious extension to DTW is to only incorporate paths in the dynamic programming that are close to the matrix's diagonal. Such a technique is called pruning.



(a) In standard DTW all paths through the dynamic programming matrix are considered.

(b) This figure illustrates DTW pruning according to the Sakoe-Chiba band. Here, only the yellow colored matrix elements are considered as relevant for the dynamic programming.

(c) The Itakura parallelogram further reduces the relevant elements by also incorporating the distance from the beginning and end in the dynamic programming.

Figure 3.3.: These figures illustrate different pruning techniques for DTW: the figure on the left shows a full dynamic programming matrix that takes all possible paths in consideration, the figure in the middle shows the Sakoe-Chiba band, and the leftmost figure displays the Itakura parallelogram. Please note that these plots illustrate the special case $T_1 = T_2$.

In order to realize pruning, a window of size r is added as an additional parameter to the DTW algorithm: r limits the distance from the diagonal in number of steps that is considered as likely paths in the dynamic programming.

Popular examples for pruning techniques are the Sakoe-Chiba band that prunes the search space according to a fixed window size r, and the Itakura parallelogram, a pruning technique that adjusts the window size in dependency of the distance to the beginning and end of the paths. See Figure 3.3 for an illustration of these pruning algorithms.

By applying these restrictions, the algorithmic complexity of DTW can efficiently be reduced from polynomial $O(T_1 \cdot T_2)$ to linear $O(T)$.

3.2.2. Hidden Markov Models

A Hidden Markov Model (HMM) can be described as a generative, discrete state space model that emits the time series O out of K hidden states.

Hidden Markov Models provide one of the most widely used techniques for analysis of time series and sequences. HMMs originate in the 1960s when the theory of HMMs were published by Baum and Petrie [1966], Baum and Eagon [1967], Baum and Sell [1968], Baum et al. [1970]. The ability to model sequential data typically arises from a combination of discrete model states with state-specific distributions of observable variables and the transitions between these states [e.g., Roweis and Ghahramani, 1999]. Successful applications of HMMs include speech recognition [Rabiner, 1989], biological sequence analysis [Eddy, 1998], and biomedical signal processing [Obermaier et al., 2001].

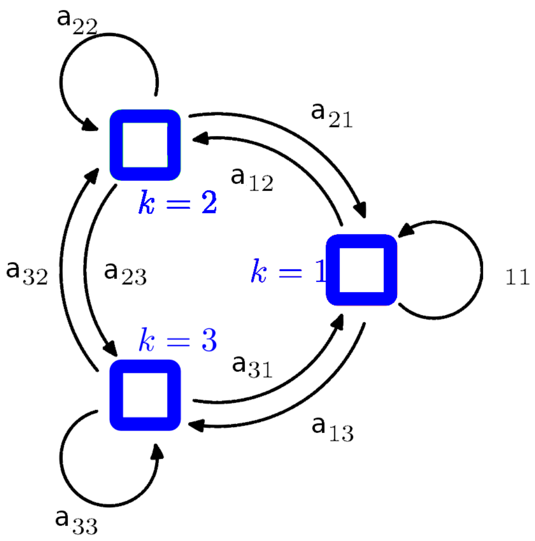


Figure 3.4.: This figure illustrates a Hidden Markov Model with three hidden states and corresponding transitions a [based on an illustration from Bishop, 2006].

Let $O = \mathbf{o}_1, \dots, \mathbf{o}_T$ be a sequence of observation vectors with $\mathbf{o}_t \in R^d$, e.g., a (multivariate) time series. The model specifies a first-order Markov chain, i.e., in a path $\mathbf{q}_T = q_1, \dots, q_T$ through the model states $q \in Q$, where the current state q_t only depends on the previous state q_{t-1}. Figure 3.4 shows an exemplary sketch of an HMM with three states.

A complete HMM λ is characterized by the tuple $\lambda = (A, B, \pi)$, with

- **State Transition Probabilities:** $A = [a_{k,l}]$ is a matrix whose elements define the state transition probabilities of the Markov chain with $a_{k,l} = P(q_t = l | q_{t-1} = k)$, $k, l = 1, \dots, K$.
- **Emissions:** $B = \{b_k(\mathbf{o_t})\}$ is a set of emission densities (or probabilities), each associated with a particular state k. These emissions can be modeled in different ways. E.g., in biological sequence analysis a common choice are discrete probabilities, whereas for speech recognition often continuous emission densities such as mixture-of-Gaussians are used.
- **Initial State Probabilities:** $\pi = \{\pi_k\}$ with $\pi_k = P(q_0 = k)$ denotes the HMM's initial state probabilities, i.e., the probabilities of starting the generating process in state k.

The application of HMMs typically requires a solution to three problems:

- **Evaluation:** Computing the posterior likelihood $p(O|\lambda)$ for an observed times series O and a given HMM λ . This problem is also associated with the question how to score an observation and how well the observation matches the given model λ.
- **Decoding:** Finding the optimal path $\mathbf{q}_T^* = q_1, \dots, q_T$ through the model λ for an observed time series O of length T.
- **Learning:** Estimating the model parameters A, B and π from an example data set $\mathbf{O} = \{O^1, \dots, O^N\}$ of observations. The model is adapted to the data set and an optimal representation is found.

The Baum-Welch algorithm — a realization of the expectation maximization algorithm — allows to train an HMM, i.e., to find a solution for the above-mentioned learning problems. The Baum-Welch algorithm estimates a parameter configuration for an HMM $\lambda = (A, B, \pi)$ by maximizing the likelihood

$$\mathcal{L} = \prod_{i=1}^{N} P(O^i|\lambda) \tag{3.2}$$

for a set of observations $\mathbf{O} = \{O^1, \cdots, O^N\}$. The algorithm incorporates an iterative optimization scheme, where

1. the emission probabilities $P(O^i|\lambda)$ of the observations for a given HMM λ are computed, and
2. the model parameters are updated according to the frequency that they are used in step 1. The resulting HMM will be an improved representation for the observations.

These two steps are repeated until convergence, i.e., until the likelihood (Eq. 3.2) converges or the model parameters do not substantially alter anymore. Rabiner [1989] gives a detailed introduction to HMMs and describes the Baum-Welch-Algorithm thoroughly.

Duration Modeling in Hidden Markov Models

A well known limitation of HMMs is that the duration to stay in one model state is geometrically distributed. The probability to stay τ time steps in one state of an HMM depends on the self-transition probability $a_{k,k}$, i.e., the probability to transit from the k-th state to the same k-th state:

$$P_k(\tau) = a_{k,k}^{\tau-1} \cdot (1 - a_{k,k}).$$

However, such a probability distribution might not be compatible to a particular distribution found in a set of observations. For example, it is known that the temporal characteristics of speech are not well represented by a geometric distribution as modeled by standard HMMs [Levinson, 1986, Rabiner, 1989, Johnson, 2005]. To circumvent this restriction, various approaches have been proposed that incorporate the state duration into the model definition. The most important approaches are:

Hidden Semi-Markov Models Hidden Semi-Markov Models (HSMMs) are similar to HMMs, but instead of emitting only one observation, each state can emit a sequence of observations. Thus, the transition probabilities in standard HMMs are replaced by explicit state duration distributions in HSMMs. Popular HSMMs are

- **Explicit Duration HMMs (EHMMs):** EHMMs explicitly model the state duration probabilities $P(\tau)$, which have to be estimated from given training data. EHMMs were introduced by Ferguson [1980].
- **Continuously Variable Duration HMMs (CVDHMMs):** CVDHMMs reduce the complexity of EHMMs by representing the duration probabilities as parametric continuous probability density functions. Here, a gamma distribution is usually used:

$$p(\tau; k, \theta) = \frac{1}{\theta}^{k} \frac{1}{\Gamma(k)} x^{k-1} e^{-\frac{x}{\theta}}, \text{ with } \Gamma(k) = (k-1)!. \tag{3.3}$$

Such a duration model only requires estimating the model parameters k and θ instead of a set of probabilities for each state. CVDHMMs were first introduced by Levinson [1986].

Variable Transition Hidden Markov Models (VTHMMs) VTHMMs model the transition probabilities $a_{i,j}(\tau)$ as a function of the elapsed duration, i.e., the number of time steps already spent in state i:

$$a_{i,j}(\tau) = P(q_i = t, q_j = t + 1|\lambda, \tau). \tag{3.4}$$

VTHMMs were introduced by Vaseghi [1995], who suggested estimating the transition probabilities from the Viterby path, i.e., the most like sequence of states.

Expanded State Hidden Markov Models (ESHMMs) ESHMMs define sub-HMMs in each state that share the same emission densities. By means of such an approach, the duration probability for a state is modeled by the duration probability of the sub-HMM. ESHMMS were introduced by Russell and Cook [1987].

However, none of these approaches reduces the overall complexity of Hidden Markov Models; instead, all models introduce further parameters, e.g., parameters of duration distributions, and/or additional states. This is reflected in higher algorithmic complexity and increased computational requirements.

3.2.3. Ordered Means Models

The duration modeling of Ordered Means Models (OMMs) also differs from standard HMMs. However, instead of explicitly defining a state duration probability and, thus, increasing the model complexity, OMMs' model design is radically reduced, which leads to implicitly changed state duration probabilities. In addition, the state duration probability $P(\tau; T)$ of OMMs depends on the length T of an evaluated time series — a so far unique approach. Please see Section 4.1.2 for a detailed analysis of OMMs' state duration probabilities.

In general, OMMs are, similar to HMMs, generative state space models that emit a sequence of observation vectors $O = \mathbf{o}_1, \ldots, \mathbf{o}_T$ out of K hidden states. However, OMMs incorporate a number of design decisions:

- **Path Probabilities:** In OMMs, each path, i.e., each valid sequence of states is equally likely. Note that such a design differs fundamentally from modeling transition probabilities in HMMs. There is no equivalent realization of equally likely paths in terms of transition probabilities (cf. Section 4.4).
- **Emission Densities:** The emissions of each state are modeled as probability distributions $b_k(\cdot)$ and are assumed to be Gaussian with $b_k(\mathbf{o}_t) = \mathcal{N}(\mathbf{o}_t; \boldsymbol{\mu}_l, \sigma)$. The standard deviation parameter σ is identical for all states and is used as a global hyperparameter.
- **Model Topology:** In OMMs, only transitions to states with equal or higher indices as compared to the current state are allowed. I.e., the network of model states follows a left-to-right topology.

- **Length Distribution:** In principle, OMMs require the definition of an explicit length distribution either by domain knowledge or by estimation from the observed lengths in the training data. This, however, may not be possible due to missing knowledge or non-representative lengths of the observations. To circumvent the definition and estimation of a length model, we assume a flat distribution in terms of an improper prior according to equally probable lengths.

With regard to these design decisions, an OMM Ω is completely defined by an ordered sequence of reference vectors $\Omega = \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ — the expectation values of emission densities.

Parameter Estimation In order to estimate particular model parameters $\Omega = \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ by a set of observations $\mathbf{O} = \{O^1, \ldots, O^N\}$ the likelihood

$$\mathcal{L} = \prod_{i=1}^{N} P(O^i|\Omega) \tag{3.5}$$

is maximized with respect to the mean vectors $\boldsymbol{\mu}_k$. To solve this optimization problem, an iterative expectation maximization algorithm similar to Baum-Welch-training of HMMs can be used.

OMMs are introduced in detail in Chapter 4.

3.3. Classification

A classification describes an arrangement of entities according to similarities, shared qualities, or features. These arrangements can be hierarchical (e.g., taxonomies as known from phylogenetic), analytical and one-dimensional, or non-hierarchical and multi-dimensional. In case of the latter, the observations are categorized according to multi-dimensional feature vectors $\mathbf{x} \in \mathbb{R}^d$.

Then, a classifier is a system that assigns previously unseen observations in form of feature vectors $\mathbf{x}$ to a category or class that is an element of a finite set of categories. A classifier can be described as a function f that maps a feature vector $\mathbf{x}$ to a class $y \in Y$:

$$f : \mathbb{R}^d \to Y, \tag{3.6}$$

with $|Y| = M, M \in \mathbb{N}$. In general, the task of classification is assigned to supervised learning.

In order to allow classification of time series data, we have to describe a function f that maps a time series from time series space $\mathbb{O}$ to a class y:

$$f : \mathbb{O} \to Y. \tag{3.7}$$

3.3.1. Bayes Classifier

Bayes classifier are a standard approaches to classification. In a Bayes classifier, the risk R of incorrect classification decisions is minimized. A loss function L has to be defined that specifies the expected costs in case of an incorrect classification for a particular task.

In order to minimize R, we then get

$$\min R = E[L] = E[L(f(x), y)]. \tag{3.8}$$

In case of Bayes classification, we assume equal costs for miss-classifications and zero costs for correct classification decisions. Given the class-specific probability density functions $p(\mathbf{x}|y^i)$ and their a-priori probabilities $P(y^i)$, we can show — based on the principles of risk minimization — that

$$P(y^i|\mathbf{x}) = \frac{p(\mathbf{x}|y^i) \cdot P(y^i)}{p(\mathbf{x})}. \tag{3.9}$$

This leads to a classification decision

$$f(\mathbf{x}) = \arg\max_{y^i} P(y^i|\mathbf{x}). \tag{3.10}$$

Since $P(y^i|\mathbf{x})$ cannot be estimated from observations directly, and the joint distribution $p(\mathbf{x})$ is identical for all classes $y^i \in Y$ we get

$$f(\mathbf{x}) = \arg\max_{y^i} \big(p(\mathbf{x}|y^i) \cdot P(y^i)\big) \tag{3.11}$$

for classification. If $p(\mathbf{x}|y^i)$ and $P(y^i)$ are unknown, the particular probabilities have to be estimated by means of adequate estimation approaches, e.g., parametric data model whose parameters are estimated from observations.

The latter approach allows us to use time series models such as HMMs or OMMs for classification of times series data. In case of OMMs the conditional likelihood $p(O|y^i)$ to observe O from class y^i is the production likelihood $p(O|\Omega^i)$ that the time series O was generated by the OMM that is associated with class y^i. Classification of time series thus implies that we first estimate one OMM Ω^i for each class $y^i \in Y$ from a given data set. If we assume equal priors $P(\mathbf{y}^i)$ for all $\mathbf{y}^i \in Y$, an unknown time series then is assigned to the class associated with the model that yields the highest production likelihood $p(O|\Omega^i)$ of all models:

$$f(O) = \arg\max_{y^i} P(y^i|O) = \arg\max_{y^i} p(O|\Omega^i). \tag{3.12}$$

3.3.2. Nearest Neighbor Classifier

Instead of estimating probability distributions for a given set of (labeled) observations, Nearest Neighbor classifiers forthrightly use the observations, i.e., the data is used without further pre-processing. Thus, Nearest Neighbor classifiers are memory-based. For classification, Nearest Neighbor classifiers assign an unknown, unlabeled observation $\mathbf{x}$ to the class that comprises the labeled observation, which is closest to $\mathbf{x}$:

$$f(\mathbf{x}) = y^i \text{ if } d(\mathbf{x}, \mathbf{x}^i) = \min_{j=1,\dots,N} d(\mathbf{x}, \mathbf{x}^j). \tag{3.13}$$

Here, $d(\cdot)$ is a distance function.

Even though $d(\cdot)$ is usually chosen to be Euclidean, it might be useful to use alternative metrics or distance functions depending on the application and the data domain.

To allow classification of time series by means of Nearest Neighbor classification, a time series distance function is required. Dynamic Time Warping as described in Section 3.2.1 provides a common choice for time series.

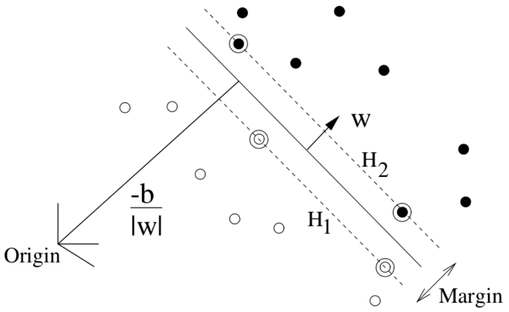


Figure 3.5.: Linear separating hyperplane in case the classes are separable. The support vectors are circled [illustration from Burges, 1998].

3.3.3. Support Vector Machines

The term Kernel Machines denotes a collection of algorithms that utilize the so-called kernel trick to implicitly map a given vector space to another, higher-dimensional vector space. One of the most popular Kernel Machines are Support Vector Machines (SVMs). SVMs are binary classifiers, targeting to distinguish two classes [Vapnik, 2000, Burges, 1998]. SVMs aim to minimize the empirical risk of an incorrect classification by means of a separating hyperplane, similar to linear classifiers such as Fisher's discriminates. Based on the statistical learning theory, the idea of SVMs is to constrain the capacity of a learned hyperplane function in order to maximize its generalization properties. SVMs are described as linear hyperplanes that are embedded in a higher-dimensional hyperspace. The mapping to this hyperspace is implicitly done by means of the kernel functions. For a detailed introduction to Support Vector Machines please refer to Burges [1998].

Linear Support Vector Machines

In the following,

$$\{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \dots, (\mathbf{x}^N, y^N)\} \in \mathbb{R}^d \times \{-1, 1\}$$

is a training set of N (labeled) observations, where the $\mathbf{x}^i$ are data vectors of dimensionality d, and y^i their corresponding class labels.

A separating hyperplane can be represented as

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3.14}$$

with a weight vector $\mathbf{w}$ and a bias term b. In order to apply such a hyperplane to prediction of an unknown example $\mathbf{x}$, the sign of the projection of $\mathbf{x}$ to the weight vector $\mathbf{w}$ reveals the estimated class label

$$\hat{y} = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b)\,. \tag{3.15}$$

In principle, there are several possible hyperplanes that separate the data set in two subsets, and the choice of one of these hyperplanes is crucial for classification performance. In

SVMs, a maximum margin approach is used as an optimization criterion for the separating hyperplane. Please see Figure 3.5 for a 2-dimensional illustration of a SVM hyperplane.

If the given examples from both classes are linear separable, the hyperplane that maximizes the margin ξ between those classes has to meet the following constraints

$$\begin{aligned} \mathbf{x}^i \cdot \mathbf{w} + b &\geq 1, \text{ if } y^i = 1 && (3.16) \\ \mathbf{x}^i \cdot \mathbf{w} + b &\leq -1, \text{ if } y^i = -1, && (3.17) \end{aligned}$$

given that the weight vector $\mathbf{w}$ is normalized. With binary class labels y^i that are either 1 or -1, these equations can be combined to

$$y^i \cdot (\mathbf{x}^i \cdot \mathbf{w} + b) \geq 1. \tag{3.18}$$

This constraint guarantees that each data point has a minimum distance of 1 to the separating hyperplane. Thus, the margin is defined as

$$\xi = \mathbf{w} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = \frac{2}{||\mathbf{w}||} \tag{3.19}$$

with $\mathbf{x}^+, \mathbf{x}^-$ representing the examples from the positive and negative classes, respectively. Maximizing the distances between all training examples and the margin ξ is, therefore, equivalent to minimizing the canonical hyperplane

$$\min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2 \tag{3.20}$$

with subject to the constraint in Equation 3.18.

Optimization In terms of optimization, a quadratic objective function (Equation 3.20) with linear constraint (Equation 3.18) has to be minimized [cf. Smola and Schölkopf, 1998]. This leads to a convex function whose minimum can be found by means of Lagrange theory [Smola and Schölkopf, 1998]. The Lagrangian function of Equation 3.20

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} ||\mathbf{w}||^2 - \sum_{i=1}^{N} \alpha_i \cdot (y^i \cdot (\mathbf{w} \cdot \mathbf{x}^i + b) - 1), \tag{3.21}$$

has to be minimized with respect to $\mathbf{w}$ and b, and maximized w.r.t. the positive Lagrange multipliers α_i. Inserting the resulting 1st derivative's constraints

$$\begin{aligned} \sum_{i=1}^{N} \alpha_i \cdot y^i &= 0, \text{and} && (3.22) \\ \mathbf{w} &= \sum_{i=1} \alpha_i \cdot y^i \cdot \mathbf{x}^i && (3.23) \end{aligned}$$

in Equation 3.21, leads to its dual representation with

$$\begin{aligned} \max_{\alpha} W(\alpha) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \cdot \alpha_j \cdot y^i \cdot y^j \cdot \mathbf{x}^i \cdot \mathbf{x}^j && (3.24) \\ \text{subject to } \alpha_i &> 0 && (3.25) \\ \text{and } \sum_{i=1}^{N} \alpha_i y^i &= 0. && (3.26) \end{aligned}$$

Given this, the decision function reveals the class label by

$$\hat{y} = f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=0}^{N} \alpha_i \cdot (\mathbf{x}^i \cdot \mathbf{x}) \cdot y^i + b\right). \tag{3.27}$$

In a geometrical interpretation (cf. Figure 3.5), all $\mathbf{x}^i$ with $\alpha_i > 0$ are called support vectors and lie on the planes H_1 and H_2. Only these vectors are relevant for the SVM solution, and given some $\alpha_i = 0$ yield a sparse representation of the hyperplane, i.e., vectors with $\alpha_i = 0$ do not contribute to the solution in Equation 3.27.

Soft Margin Approach

However, the assumption of linear separable data does only very rarely match what is required in real-world data. In order to allow violation of the constraint in Equation 3.18, so-called slack variables ξ^i are added:

$$y^i \cdot (\mathbf{w} \cdot \mathbf{x}^i + b) \geq 1 - \xi^i, \text{with } \xi \geq 1, \forall i = 1, \ldots, N. \tag{3.28}$$

By introducing a penalty term to the objective functions

$$\min_{\mathbf{w},\xi} \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi^i \tag{3.29}$$

small ξ values are preferred during minimization. C is a regulation of the penalty term, and higher values of C correspond to a stronger penalization. According to the dual representation of the objective function, only an upper bound is added to the constraint in Equation 3.25

$$0 \leq \alpha_i \leq C. \tag{3.30}$$

The penalty regulation parameter C has to be chosen by means of a hyperparameter selection algorithm, e.g., $\mathcal{N}$-fold cross validation (cf. Section 5.2.1).

Non-linear Support Vector Machines An alternative approach to separate data that are not linear separable, is to transform the data to another feature space $\mathbb{H}$ in which the data is linear separable. The data set is transformed

$$\Phi : \mathbb{R}^d \to \mathbb{R}^D, \tag{3.31}$$

to a, usually, higher-dimensional feature space $\mathbb{R}^D$ $(d < D)$. For this, the expression $\mathbf{x}^i \cdot \mathbf{x}^j$ is replaced by $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^j)$ in Equation 3.27.

Kernel Trick However, choosing a suitable transformation Φ is a non-trivial task that comes at additional cost and, possibly, requires additional parameter selection. Instead of explicitly defining a mapping function Φ, the so-called kernel trick is capable to implicitly map the data to another feature space while dramatically reducing the computational costs.

The scalar products in Equations 3.27 are replaced by a kernel functions $k(\cdot)$ with

$$\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^j) = k(\mathbf{x}^i, \mathbf{x}^j). \tag{3.32}$$

Necessary prerequisites for functions to be used as a kernel functions $k(\cdot)$ are

- **Symmetry:** $k(\mathbf{x}^i, \mathbf{x}^j) = k(\mathbf{x}^j, \mathbf{x}^i)$,
- **Cauchy-Schwarz Inequality:** $k(\mathbf{x}^i, \mathbf{x}^j)^2 = \left(\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^j)\right)^2 \leq ||\Phi(\mathbf{x}^i)||^2 \cdot ||\Phi(\mathbf{x}^j)||^2$.

A sufficient precondition is Mercer's Condition that requires a positive semi-definite $N \times N$ Kernel matrix (often called Gram matrix) $K_{i,j} = k(\mathbf{x}^i, \mathbf{x}^j)$.

Among others, established kernel function are

- **Linear Kernel:** $k(\mathbf{x}^i, \mathbf{x}^j) = \mathbf{x}^i \cdot \mathbf{x}^j$,
- **Polynomial Kernel:** $k(\mathbf{x}^i, \mathbf{x}^j) = (\mathbf{x}^i \cdot \mathbf{x}^j)^l$,
- **Radial Basis Function Kernel:** $k(\mathbf{x}^i, \mathbf{x}^j) = \exp\left(-\gamma \cdot ||\mathbf{x}^i \cdot \mathbf{x}^j||^2\right)$,
- **Neural Net Kernel:** $k(\mathbf{x}^i, \mathbf{x}^j) = \tanh\left(\kappa \mathbf{x}^i \cdot \mathbf{x}^j - \delta\right)$.

Again, choosing the kernel parameters γ, l, κ, etc. requires an external parameter selection process.

Last, it is possible to combine both approaches for separation of data that are not linear separable — an additional penalty term and an implicit data transformation into another feature space by means of the kernel trick. However, this induces the selection of at least two hyperparameters what comes at additional computational costs.

Time Series Kernels To use kernel machines such as SVMs for analysis of time series, various approaches exist. On one hand, time series data can be embedded in a traditional feature space by applying a feature extraction to the time series. However, more advance approaches utilize the kernel trick by defining specific time series kernels. Time series kernels apply a kernel function $k(O^i, O^j)$ to two time series O^i, O^j that maps from time series space $\mathbb{O}$ to real numbers $\mathbb{R}$: $k : \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{R}$.

Interpolation The probably simplest approach to time series kernels is an interpolation of the time axis. In context of speech recognition, Clarkson and Moreno [1999] suggest to apply a linear normalization in time, i.e., time series of different length are interpolated to equal length while the kernel function is applied. After such a pre-processing, standard vector space metrics can be used to further process the interpolated series in a SVM.

Fisher Kernel Another family of time series kernels employ probabilistic models such as HMMs in kernel functions. E.g., the Fisher kernel [cf. Jaakkola et al., 1999] uses the fisher score

$$U_O = \nabla_\theta \log P(O, \theta)$$

with O denoting one time series, and $\theta \in \Theta$ being one particular parametrization of a generative model. The Fisher kernel is then defined as

$$k_{fisher}(O^i, O^j) = U_{O^i}^T U_{O^j}.$$

Fisher kernels combine the advantages of generative models with discriminative approaches such as SVMs. Originating in bio-informatics, Fisher kernels have been successfully applied to various data domains, e.g., audio mining [Moreno and Rifkin, 2000] and speaker identification [Wan and Renals, 2002]. A similar approach, the conditional symmetric independence kernel, proposed by Watkins [1999], incorporates Pair Hidden Markov Models for computation of the kernel values.

Alignment Kernel A different approach for time series kernels are Alignment kernels. Such kernel functions utilize a dynamic-time alignment such as DTW or edit-distance (cf. Section 3.2) for computation of the kernel values. These kernels use the alignment scores as a similarity or distance function.

A simple alignment kernel is proposed by Gudmundsson et al. [2008]. They suggest to choose the negative dynamic time warping distances as kernel (cf. Section 3.2.1) values:

$$k_{ndtw}(O^i, O^j) = -d_{dtw}(O^i, O^j).$$

In addition, Bahlmann et al. [2002] define the Gaussian-Dynamic-Time-Warping kernel (GDTW) straight forward, replacing the Euclidean distances in RBF kernels (cf. Section 3.3.3) with DTW distance values:

$$k_{gdtw}(O^i, O^j) = \exp\left(-\lambda \cdot d_{dtw}(O^i, O^j)\right).$$

Similarly, the authors of Shimodaira et al. suggest an approach where the “DTW distance” is replaced by a “DTW score” with a likewise exponentiation.

However, note that none of the so far presented alignment kernel functions fulfill Mercer’s Condition. Thus, none of these approaches does provide a valid kernel function. In contrast, Cuturi et al. [2007] proposed an approach in which all time-alignment paths are considered for computation of the kernel value, and the authors claim that their alignment kernel is positive semi-definite.

Support Vector Machines for Multi-Class Classification

So far, SVMs are only used for binary classification, i.e., classification of two classes. However, most classification problems involve more than two classes ($M > 2$). In order to apply binary SVM classifiers to multi-class problems, a given multi-class set has to be broken down to a series of binary classifications problems [e.g., Chang and Lin, 2011]. A strategy to process M-class problem is obtained by M binary classifiers, each of which is trained with the data of one class as elements of the positive class, while the examples from the remaining classes are used as negative training data. An alternative approach is to use $M(M-1)/2$ binary classifiers, each trained with the data of one class against another class. The first approach is called one-against-all, the second one-against-one training.

3.4. Clustering

Clustering is a fundamental technique for data analysis and data mining. Its basic idea is to discover hidden structures and similarities in data collection without prior knowledge, i.e., in an unsupervised manner. Clustering has been widely used and studied for at least 5 decades, in which many methods and applications have been developed. For a good overview, please refer to Berkhin [2006]. Most popular clustering methods are either hierarchical or partitioning clustering approaches. While the first ones try to detect hierarchical, refining similarity structures in a set of observations, the latter partition the data set into K groups according to a similarity or distance measure and an objective function.

In order to allow clustering of time series data, the algorithms need to reflect particular properties of the data that are often connected to data values that change in time, i.e., varying length and compression or expansion of the time-axes. According to Liao [2005], we can distinguish between three clustering approaches for time series data: (i) approaches based on raw data observations, (ii) clustering based on feature representation, and (iii) model-based clustering approaches. The particular algorithms differ in how the observations and the discovered clusters are represented. The major focus of this thesis is on model-based clustering solutions, thus, clusters are represented by data model. A frequently apply model for clustering of time series data are HMMs. E.g., Alon et al. [2003] and Smyth [1997] propose algorithms based on mixture-of-HMMs with either probabilistic or deterministic; Oates et al. [1999] apply an iterative procedure in which a set of HMMs is found, with each HMM accepting a disjoint subset of the original set of sequences. In contrast to approaches that use models to represent clusters, spectral clustering algorithms apply time series kernels to time the observations and process gram matrices [e.g., Jebara et al., 2004, 2007].

3.4.1. $\mathcal{K}$-means

The general idea of $\mathcal{K}$-means clustering is to minimize an error function that accumulates the distances between each observation and a prototype, often called cluster centroid or cluster means, to which it is assigned to. The algorithm that minimizes the error proceeds in two steps: first, each observation is assigned to its nearest prototype according to a distance measure. Second, each prototype is updated with respect to the assignments. These two steps are repeated until a termination condition is reached, e.g., the error function is converged.

$\mathcal{K}$-means clustering is a common method to partition a set of observations into $\mathcal{K}$ groups. It was initially developed more then 40 years ago [MacQueen and Others, 1966]. The general idea of $\mathcal{K}$-means clustering is to minimize an objective function that accumulates the distances between each observation $\mathbf{x}^i$ and a prototype $\mathbf{c}^j$ (often called "cluster centroid" or "cluster means") to which it is assigned to, expressed as

$$\min E = \sum_{i=1}^{N} \sum_{j=1}^{\mathcal{K}} w_{i,j} \cdot d(\mathbf{x}^i, \mathbf{c}^j)^2 \tag{3.33}$$

subject to $w_{i,j} \in \{0, 1\}$ and $\forall j : \sum_{i=1}^{\mathcal{K}} w_{i,j} = 1$. The latter constraint assures that each example is only assigned to one prototype. The basic algorithm that minimizes this error proceeds as follows:

First, assign each observation $\mathbf{x}^i$ to its closest prototype $\mathbf{c}^j$ according to a distance measure $d(\mathbf{x}, \mathbf{c})$. Second, update the prototypes $\mathbf{c}^1, \dots, \mathbf{c}^{\mathcal{K}}$ according to their assigned observations. These steps are repeated until convergence.

For observations that are embedded into a d-dimensional feature space, i.e. $\mathbf{x}^i, \mathbf{c}^j \in \mathbb{R}^d$, $d(\mathbf{x}^i, \mathbf{c}^j)$ usually is the Euclidean distance measure and the function that updates the prototypes is the average function. However, depending on the application, other distance measures and update functions might be suitable as well. If the observations are time series, such a distance measure is impractical. One way to overcome this issue is to use generative probabilistic models such as HMMs. To apply these models to clustering analysis, a common distance function between a time series O^i and a probabilistic model Ω^j is the negative production log-likelihood $d(O^i, \Omega^j) = -\ln p(O^i|\Omega^j)$. By this, the general objective function in Equation 3.33 accumulates the squared log-likelihoods that the time series O^i has been generated by the centroid models. The update function in such an approach is a re-training of the models according to the assigned time series [Smyth, 1997, Oates et al., 1999, Perrone and Connell, 2000, Alon et al., 2003].

Due to local extrema, the quality of the $\mathcal{K}$-means solution heavily depends on initialization. A naïve and yet common approach is to randomly choose initial prototypes or clusters as initialization for the first iteration. Other authors [e.g., Smyth, 1997] suggest an initialization based on pairwise distances between the examples of a data set. These distances then can be exploited to hierarchically clustering of the data set and encourage compact cluster initializations beforehand.

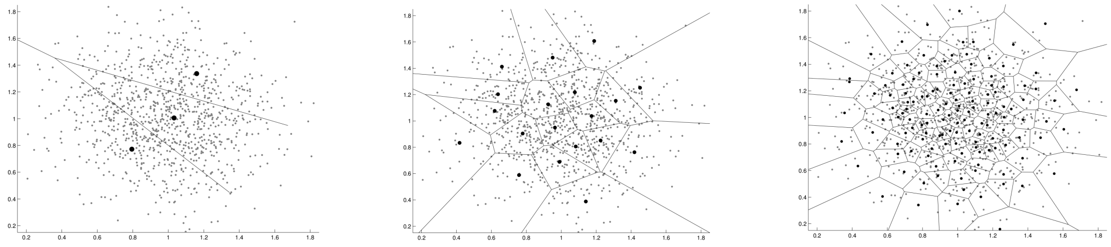
3.4.2. $\mathcal{K}$-trees

For very large data sets, common variations of the $\mathcal{K}$-means algorithm involve efficient search procedures, usually based on search trees. A powerful approach is the $\mathcal{K}$-tree algorithm [Geva, 2000, De Vries and Geva, 2009a,b], which is a hybrid of the before-mentioned $\mathcal{K}$-means algorithm and classical B+-trees [Comer, 1979]. The $\mathcal{K}$-tree algorithm constructs a balanced search tree, where each node contains a key that is used for comparison, and each leaf holds an observation. While achieving similar results as compared to standard $\mathcal{K}$-means, the training procedure of $\mathcal{K}$-trees is computationally more efficient and, thus, scales up to much larger data sets. In contrast to $\mathcal{K}$-means clustering, the $\mathcal{K}$-tree algorithm does not require that the number of cluster centroids to be specified beforehand.

A $\mathcal{K}$-tree provides a nearest neighbor search tree for data that is represented as real-value vectors $\mathbf{x} \in \mathbb{R}^d$. The tree's nodes n are represented as an array of tuples $(\mathbf{v}, c)$, where each $\mathbf{v}$ is a vector $\mathbf{v} \in \mathbb{R}^d$, and each c is a child node. The number of elements l of the array n cannot exceed the tree's order m:

$$1 \leq l \leq m.$$

In case that a node is a leaf, the child c is empty. The vector $\mathbf{v}$ is the mean of all sub-adjacent leaves, i.e., the vectors $\mathbf{x}$ that represent the observations. A $\mathcal{K}$-tree is build bottom-up and a full node, i.e., a node whose length l is equal to the tree's order m, is split by applying $\mathcal{K}$-means to its vectors $\mathbf{v}$.



(a) Level 1 (b) Level 2 (c) Level 3

Figure 3.6.: Voronoi cells on according to a data partitions of a $\mathcal{K}$-tree on different levels. The data were drawn randomly from a Gaussian distribution with a location parameter of 1, and a deviation parameter of 0.3. The $\mathcal{K}$-trees order was $m = 11$ [illustrations from De Vries, 2010].

A $\mathcal{K}$-tree of order m is defined as [c.f. Geva, 2000]:

1. All leaves are on the same level.
2. All internal nodes, including the root, have at most m non-empty children, and at least one nonempty child.
3. Codebook vectors act as search keys.
4. The number of keys in each internal node is equal to the number of its non-empty children, and these keys partition the keys in the children to form a nearest neighbor search tree.
5. Leaf nodes contain data vectors.

Since a $\mathcal{K}$-tree is ultimately a nearest neighbor search tree, the time complexity of building a $\mathcal{K}$-tree is $O(N \cdot \log N)$ where N is the number of observations. The low time-complexity of $\mathcal{K}$-trees originates in the efficiency of the $B+$-algorithm.

$\mathcal{K}$-trees have been used to analyze very large data sets, usually, for the purpose of information retrieval and document clustering. Here, as, e.g., shown by De Vries and Geva [2009a,b], the $\mathcal{K}$-tree algorithm provides results of similar quality as standard clustering tools that are popular in information retrieval [e.g., CLUTO, Karypis, 2003], while offering substantially decreased run-time requirements.

Building a $\mathcal{K}$-tree Building a $\mathcal{K}$-tree is a dynamic process that inserts data vectors on-line, as they arrive. Consider an already existing $\mathcal{K}$-tree of order m, and an additional vector $\mathbf{x}$ that is presented to insertion. The insertion procedure is: first, the tree is searched to identify the node that contains the vectors' nearest neighbor. If this node stores less than m — the tree's order — vectors, $\mathbf{x}$ is inserted into this leaf. Since this insertion changes the cluster structure, the parent nodes up to the root node must update their search keys in terms or re-computing the means.

When a leaf stores $m + 1$ vectors, it can not contain any more elements. It then is split by applying the $\mathcal{K}$-means algorithm with $k = 2$ to the $m + 1$ elements of that node. The resulting centroids become the search keys for two new child nodes, each containing the associated vectors. Now consider that the parent of these new nodes contains $m + 1$ search keys after the splitting process. Then it also has to be split, replacing itself in its parent by the two new child nodes. This process is repeated in a similar manner until a node with less than m search keys is reached, or the root node is replaced by a new root. The procedure of building a $\mathcal{K}$-tree is initialized with an empty root node and just one leaf.

In order to use $\mathcal{K}$-trees for clustering of time series data, we will propose an approach based on OMMs in Chapter 8.

3.5. Application in Interaction Scenarios

Classification and clustering techniques are often used to recognize, reproduce, and organize interaction data. Among others, handwriting, speech, and gestures are popular research topics that created successful applications in the past. Others, such as Brain Computer Interfaces translate brain activity — for instance recorded as readings from Electroencephalography devices — in commands for computers, and gaze estimation systems measure an individual's visual attention by tracking the location where a human user is looking in 3D.

3.5.1. Recognition

Recognition of interaction data has a long history in computer science research. In particular, handwriting, speech, and gesture recognition are popular research topics for decades that already yielded many applications.

Recognition of handwriting has been subject to pattern recognition and image processing research for a long time. Depending on the recognition task, i.e., online or offline, different recognition algorithms can be used. Offline recognition of handwritten text allows comprehensive data pre-processing and feature extraction before applying a classifier in the actual recognition step. Thus, the application of any classification algorithm with standard feature space representation is possible. Such approaches include Neural Networks [e.g., Graves et al., 2009, Arora et al., 2010, Asthana et al., 2011], Support Vector Machines [Bin et al., 2000, Bahlmann et al., 2002], Nearest Neighbor classifiers [Bottou et al., 1994], and others [for an overview, please see Liu et al., 2002]. If the handwriting is represented as time series, particular time series classifiers are used that employ an observation alignment, e.g., Dynamic Time Warping or Hidden Markov Models. Here, in particular Hidden Markov Models are a widely used choice [Bercu and Lorette, 1993, Hu et al., 1996, Schenkel et al., 1995, El-Yacoubi et al., 1999, Marti and Bunke, 2001, Plötz and Fink, 2009]. Other authors suggest hybrid classifiers, e.g., combinations of Neural Networks and Hidden Markov Models [Bourlard and Morgan, 1994].

Automatic speech recognition (ASR) describes the process of translating spoken words into text by means of an autonomous computer system. ASR has been studied for many decades and its performance improved significantly over the years. Today, commercial solutions for ASR, for example, in interactive voice response systems or reading voice dictation, exist. Most of the ASR systems utilize statistical modeling techniques for speech recognition, with Hidden Markov Models being one of the most successful modeling approaches. In this context, HMMs are usually used to represent sub-words, e.g., phonemes or syllables [Rabiner, 1989, Juang and Rabiner, 1991]. A well-known limitation of HMMs is that the state duration probabilities are geometrically distributed, which does not well represent the temporal characteristics of speech [Levinson, 1986, Johnson, 2005]. To circumvent this restriction, various approaches have been proposed that incorporate the state duration into the model definition [Ferguson, 1980, Levinson, 1986, Vaseghi, 1995]. Other authors suggest to use discriminative training strategies for HMMs to improve the recognition performance [Nádas et al., 1988, Juang and Katagiri, 1992, Juang et al., 1997, Katagiri et al., 1998, McDermott et al., 2007], and again others utilize Sparse Kernel Machines such as Support Vector Machines for ASR [Ganapathiraju et al., 2004, Hamaker et al., 2002, Smith and Niranjan, 2001]. However, all of these ASR approaches have in common that extensive data pre-processing is necessary to extract relevant information from the data.

A third widely used interaction modality is gestural interaction, and various approaches for gestures recognition exists. Technically, these approaches usually include (i) sensing technologies, e.g., the use of tracking or vision devices, (ii) data pre-processing, i.e., feature extraction, normalization, etc. and (iii) recognition algorithms, i.e., statistical modeling, image processing, etc. Popular choices for gesture recognition algorithms are HMMs [Yamato et al., 1992, Samaria and Young, 1994, Starner and Pentland, 1995, Starner et al., 1998, Yoon et al., 2001, Bowden et al., 2003, 2004] and Finite State Machines [Davis and Shah, 1994, Bobick and Wilson, 1997, Yeasin and Chaudhuri, 2000, Hong et al., 2000], but also Kalman and Particle Filtering approaches [Black and Jepson, 1998, Maskell and Gordon, 2001, Ramamoorthy et al., 2003, Kwok et al., 2004], Neural Networks such as Multilayer Perceptrons or Time-Delay Neural Networks [Sung and Poggio, 1995, Rosenblum et al., 1996, Yang and Ahuja, 1999], or Dynamic Time Warping [Gavrila et al., 1995, Akl and Valaee, 2010] are used.

Brain Computer Interfaces translate brain activity in commands for computer devices. Brain activity is usually measured by means of electroencephalography (EEG) or functional magnetic resonance imaging (FMRI) recordings and machine learning algorithms are used to learn the relevant characteristics of the data. Various learning algorithms are used, ranging from Hidden Markov Models [Obermaier et al., 2001], Support Vector Machines [Kaper et al., 2004], to simpler linear separating algorithms [Finke et al., 2009].

3.5.2. Reproduction

Speech generation, also known as speech synthesis, is, similar to speech recognition, a major research field in computer science for decades. There exist various approaches to synthesize speech: The simplest and yet widely used approach is concatenative synthesis where pre-recorded speech segments, e.g., words, are concatenated to build complete sentences. In contrast, in unit selection synthesis“units”, i.e., realizations of phones, diphones, etc., are selected from large databases. In these databases additional information such as frequency, pitch, neighborhood phones, etc. are stored, and a desired unit is selected according to these information, usually, by means of search trees [Hunt and Black, 1996, Conkie, 1999]. A completely different approach is statistical parametric synthesis. Here, generative models, mostly a combination of multiple Hidden Markov Models, are used to synthesize speech parameter sequences [Yoshimura et al., 1999, Tokuda et al., 2000, 2002, Black et al., 2007].

Gesture reproduction and generation is usually engineered as an alignment task with speech generation. Such approaches have shown promising results, in particular in Embodied

Conversational Agents research [Yan, 2000, Salem et al., 2012]. However, if gesture generation is examined independently of other modalities, the most used approaches are based on generative models, particularly Hidden Markov Models. Inamura et al. [2003] use a straightforward generation process with HMMs trained for joint angle trajectories of human motion observations: new trajectories are generated by simulating state transitions and emissions, with averaged repeated trails. In contrast, Calinon and Billard [2004] employ HMMs in imitation learning scenarios with the aim to learn generalized gesture representations from multiple observations. They generate new gestures by interpolating mean vectors from the observations [see also Calinon and Billard, 2005, 2007, Asfour et al., 2008, Kwon and Park, 2008].

3.5.3. Organization

Organization and prototyping of interaction data is usually done to gather a condensed data representation that allows (i) data quantization, or (ii) performance prototyping. Organization of interaction data requires usually unsupervised machine learning algorithms such as $\mathcal{K}$-means clustering or Self-Organizing Maps.

Relating to handwriting, Schomaker [1993] uses Self-Organizing Maps to organize strokes, i.e., pieces of handwriting movement bounded by minima in the pen-tip velocity, and characters for feature vector quantization. The obtained prototypes that represent the neurons can then be interpreted as abstract character (or stroke) representations [cf. Figure 3 on page 448 in Schomaker, 1993]. In contrast, Kurniawan et al. [2009] use Self-Organizing Maps to discover characters/segments in cursive handwritten text. Hierarchical clustering algorithms for handwriting are used in different contexts: Jay et al. [2001] apply clustering algorithms to pre-sort character segments to allow data-driven design of HMM topologies; Vuori et al. [2001] use clustering in order to condense a training data set; while Vuurpijl and Schomaker [1997] use clustering techniques to categorize large data sets of handwriting in hierarchical structures. Similarly, Perrone and Connell [2000] use a $\mathcal{K}$-means clustering approach to extract sub-class models in an unsupervised manner.

In both, ASR and speech synthesis systems, clustering algorithms are used to pre-select relevant elements of a given speech data set, or organize large data sets to allow quick data retrieval in a selection process [Nakajima and Hamada, 1988, Black and Taylor, 1997].

There also exists research towards organization of gestures. Most of these approaches share that they rely on a clustering algorithm to discover hidden structures in gesture observation sets. For example, Heidemann et al. [2004] use Self-Organizing Maps to automatically cluster pointing gestures in order to simplify data labeling for a, subsequently, supervised training of a gesture classifier. Similarly, Steffen et al. [2007, 2008], Steffen [2010] apply Self-Organizing Maps and Unsupervised Kernel Regression to structure grasping and manipulative motions that allow robots to perform adequate grasping motions as required in a particular situation. Other authors suggest using $\mathcal{K}$-means algorithms for clustering [e.g., Wachs et al., 2005] or quantization [e.g., Schlömer et al., 2008] of gesture data. Additionally, clustering with distance functions based on HMMs [e.g., Wang et al., 2001] are proposed. Some of these organization approaches also use the obtained clusters as prototypes, either for quantization [Schlömer et al., 2008] or as performance templates [Steffen, 2010].

		predicted outcome		
		p	**n**	**total**
actual value	**p′**	True positive	False negative	P′
	n′	False positive	True negative	N′
	total	P	N	

Figure 3.7.: This figure illustrates a confusion matrix with four possible outcomes of a binary classifier. The columns denote the actual classification value (the label) and the rows refer to the predicted outcome.

3.6. Performance Measures

3.6.1. Classification

A classifier maps observations to a finite set of classes. In order to rate the performance of a classifier or to compare classifiers against each other, we are interested in the relation of correctly vs. incorrectly classified observations. However, depending on the aim of an application, we might need a more detailed analysis of the classification process. E.g., if a classifier is used to monitor an industrial plant, it might be catastrophic if the classifier does not alert a potentially life-threatening event to the operators due to an incorrect classification decision. In such a case, the *costs* of a *false negative* classification might be enormous — and a system operator would welcome a large amount of false positive alarms, if a real emergency is rightfully notified.

Consider a binary classifier whose classification decisions can be either positive or negative. There are four possible outcomes from such a classifier [Fawcett, 2006]:

- **True Positive (TP):** When the classification yields a positive value and the actual value is indeed positive.
- **True Negative (TN):** When the outcome of the classification is negative and the actual value is negative.
- **False Positive (FP):** When the classification yields a positive value, but the actual value is negative.
- **False Negative (FN):** When the outcome of the classifier is negative but the actual value is positive.

If we take a classification task with P positive and N negative observations, these rates are usually illustrated by means of a confusion matrix (cf. Figure 3.7). However, to quantitatively assess the results of a classification outcome, the elements of the confusion matrix can be used to define measures that rate the performance of the classifier:

- **Sensitivity** (TPR) (in other contexts also called true positive rate) describes the rate of correctly classified positive examples in all available positive examples: $TPR = TP/P$.
- **Specificity** (TNR) (sometimes called true negative rate) analogously denotes the number of correctly classified negative examples in relation to the overall number of negative examples.
- **Accuracy** (ACC) denotes the proportion of true classification outcomes, i.e., the correctly classified positive and negative examples: $ACC = (TP + TN)/(P + N)$.

3.6.2. Generalization Error

In this thesis, we will use a common evaluation scheme to empirically rate the performance of classification algorithms: the generalization error. A classifier's property to recognize previously unseen observations is called generalization. To ensure good generalization performance, it is necessary to avoid over-fitting, i.e., to make sure that the classifier represents an underlying dependency, instead of learning minor changes in the data. In this thesis, we will apply the generalization error GE as a performance measure for classifiers. The generalization error describes the expected rate of incorrect classifications that occur in unseen data:

$$GE = E[L(f(\mathbf{o}^i), y^i)],$$

which is easy estimated by the rate of incorrectly classified examples in relation to the absolute number of classified examples in a test set with N examples:

$$\widehat{GE} = \frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{o}^i), y^i).$$

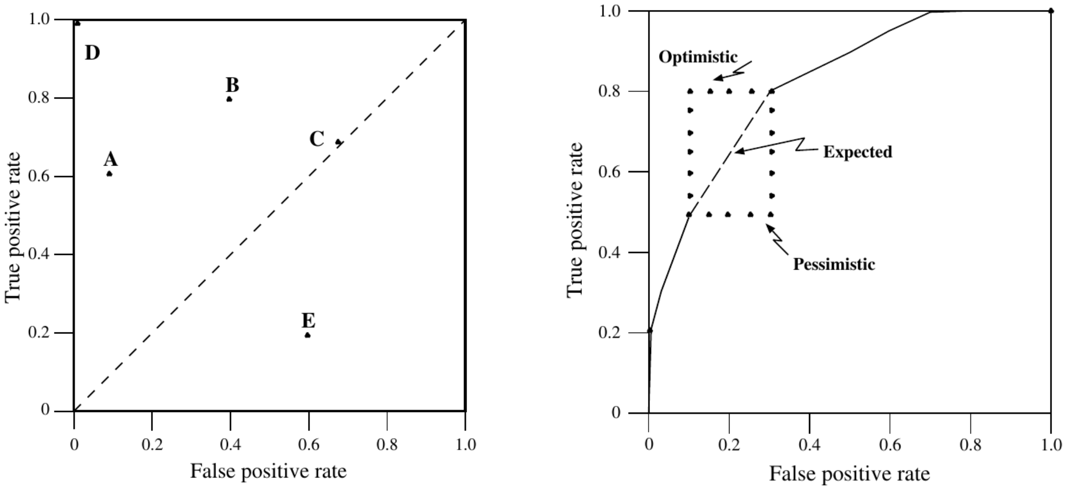
Here, L denotes the loss function (cf. Sec. 3.3.1).

3.6.3. Receiver Operating Characteristic

However, the generalization error measure is prone to skewed and dynamic class distributions, in particular, in scenarios with unequal classification costs. E.g., Fawcett [2006] gives an example from the domain of medical decision making where the class distribution changes due to an epidemic disease. I addition, scenarios with only few positive and a large amount of negative examples are common, e.g., in failure prediction for production facilities.

Receiver Operating Characteristics (ROC) analysis provides an approach to circumvent the above-sketched shortcomings of generalization error analysis. ROC graphs are a technique for visualizing classifier performances in terms of exploding the proportion of TP- and FP-rate of a classification system. This implies that a classifier is not rated by an actual (mis-)classification rate but by the proportion of incorrectly classified positive examples in relation to mistakenly positive classified negative examples, i.e., the relation of sensitivity and 1−specificity.

A ROC graph shows the *true positive rate* on the x-, and the *false positive rate* on the y-axis of a plot. Figure 3.8a illustrates this exemplary for five hypothetical classifiers.



(a) A Receiver Operating Characteristic graph. The x-axis represents the *true positive rate*, while the y-axis the *false positive rate.*

(b) A Receiver Operating Characteristic curve that is obtained by varying the classification threshold from $-\infty$ to ∞ and tracing the ROC values for the classifiers.

Figure 3.8.: These figures illustrate Receiver Operating Characteristic graphs. Both illustrations are from Fawcett [2006].

Classifiers that are represented by points in the lower-right triangle (below the diagonal in Fig. 3.8a) perform worth than classifiers with random classification decision, whereas points in the upper left triangle of a ROC graph represent classifiers that perform better than random. A perfect classifier is located in $(0, 1)$, and a point in $(0, 0)$ is related to a classifier that never yields a positive classification decision — in opposite to a classifier in $(1, 1)$ that only yields positive classification predictions.

Fawcett [2006] informally characterizes the ROC space by entitling points on the left-hand side close to the x-axis as “conservative” since they only issue positive classifications “with strong evidence so they make few false positive errors, but they often have low true positive rates as well” [Fawcett, 2006, p. 863].

Points on the upper right-hand side of a ROC graph are “liberal” according to Fawcett [2006]: “they make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates” [Fawcett, 2006, p. 863].

Curves in ROC space If a classifier yields a probability or a score that is related to the degree of which an observation is a member of a class, a common classification approach is to define a threshold value. An observation for which a classifier yields a probability or score value below the defined threshold will be classified as negative, while an observation with a probability or score above the threshold will be classified as positive. Since in dependency of the threshold the point in ROC space differs it is possible to vary the threshold from $-\infty$ to ∞ and trace its ROC values through ROC-space. The evolving path of changing TP- and FP-rates is called ROC curve. Please note that most classifiers can be re-defined in a way that they issue a score that describes the classifier’s certainty, and thus, qualify for ROC curve analysis. Figure 3.8b shows an example ROC curve for a hypothetical classifier.

Area Under Curve A ROC curve is a 2-dimensional representation of a classifier's performance. In order to reduce this visual presentation to a quantitative value that is easily interpretable, a common approach is to calculate the *Area under Curve (AUC)*. The AUC value lies between 0 and 1, with a value of 0.5 indicating a random classification. Fawcett [2006] points out that the "AUC has an important statistical property: the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance."

In order to extend AUC analysis to multi-class problems, we calculate the average $\text{AUC}_{\text{total}}$ value weighted by the class' prior probability:

$$\text{AUC}_{\text{total}} = \sum_{y^i \in Y} \text{AUC}(y^i) \cdot p(y^i)$$

where Y denotes the set of classes and $p(y^i)$ the class's y^i prior probability [Provost and Domingos, 2003].

For a detailed introduction to Receiver Operating Characteristics, please see the introduction by Fawcett [2006].

3.6.4. Clustering

To evaluate the quality of an unsupervised clustering, we can distinguish between intrinsic and extrinsic evaluation approaches.

Intrinsic evaluation is based on inter- and intra-cluster distances or similarities, i.e., how close the examples that are assigned to the clusters are located to each other, and how distant they are from elements of other clusters. Extrinsic approaches incorporate ground truth to rate the clustering solution. This induces that all data are labeled with the associated correct cluster. Based on this information, the estimated clustering solution can be compared to the aspired correct solution. In the following section, we will use the nomenclature suggested by Rosenberg and Hirschberg [2007]: the known ground truth clustering is denoted by a set of classes $Y = \{y^i | i = 1 \dots n\}$, and the learned clustering is represented by a set of clusters $C = \{c^i | i = 1 \dots m\}$. In addition, $A = [a_{i,j}]$ is a matrix produced by the clustering algorithm. $a_{i,j}$ is the number of vectors that are elements of class y^i and that are associated to cluster c^j. Extrinsic evaluation measures are still subject to discussion in the research community and many approaches to measure the distance between ground truth and an estimated clustering exist. Amigó et al. [2009] distinguish four extrinsic clustering measures:

- **Evaluation by set matching:** These measures rely on specificity and sensitivity from classification evaluation (cf. 3.6.1). According to Amigó et al. [2009], these set matching evaluation measures "share the feature of assuming an one to one mapping between clusters and categories". Purity [Zhao and Karypis, 2001] and F-measure [Steinbach et al., 2000] are examples for set matching evaluation measures.
- **Metrics based on counting pairs:** These metrics incorporate statistics about pairs of elements [Halkidi et al., 2001, Meila, 2005]. The metrics usually compare if these pairs share classes in ground truth and the estimated clusters, e.g., if pairs

of elements that are in the same class are also in the same learned cluster. Popular metrics from this family are Rand R or Jaccard Coefficient J.

- **Metrics based on entropy:** These metrics are motivated by information theory. The entropy of a cluster measures how the ground truth classes are distributed in the learned data clustering. Metrics based on entropy are, e.g., mutual information [Dom, 2002], Q_0 [Xu et al., 2003], or V-measure [Rosenberg and Hirschberg, 2007].
- **Metrics based on edit distance:** Here, a clustering solution is evaluated by the number of transformation steps that are necessary to obtain the original clustering [Bagga and Baldwin, 1998].

For a comprehensive comparison and detailed descriptions of external classification measures, please refer to Amigó et al. [2009].

The work described in this thesis incorporates four extrinsic clustering measures: purity, homogeneity, completeness, V-measure:

- **Purity:** purity implies that a perfect clustering solution consists of clusters that only contain examples from a single class. Imperfect clustering solutions deviate from this, i.e., not all but still a majority of data examples from a learned clustering should be associated with a class from ground truth. In this sense, purity is similar to the accuracy measure from classification, but instead of given class labels, the clusters are assigned to a category by majority voting. According to the original authors [Zhao and Karypis, 2001] who described purity in the first place, it “measures the extent to which each cluster contained documents from primarily one class”. Purity is defined by

$$purity = \sum_{r=1}^{m} \frac{1}{n} \max(n_r^i)$$

where m is the number of clusters, n_r is the size of cluster r, and n_r^i is the number of data vectors in class y_i that is assigned to the learned cluster r.

- **Homogeneity:** A similar approach is homogeneity, which prefers clustering results that contain data vectors from one class. This measure examines the conditional distribution $H(Y|C)$ normalized by information provided by the clustering $H(Y)$. With

$$\begin{aligned} H(Y|C) &= -\sum_{c=1}^{m}\sum_{y=1}^{n} \frac{a_{y,c}}{N} \log \frac{a_{y,c}}{\sum_{i=1}^{n} a_{i,c}} && (3.34) \\ H(Y) &= -\sum_{y=1}^{n} \frac{\sum_{c=1}^{m} a_{y,c}}{n} \log \frac{\sum_{c=1}^{m} a_{y,c}}{n} && (3.35) \end{aligned}$$

the homogeneity h is then defined as

$$h = \begin{cases} 1 & \text{if } H(Y|C) = 0, \\ 1 - \frac{H(Y|C)}{H(Y)} & \text{else.} \end{cases} \qquad (3.36)$$

The inversion of 1 and 0 is introduced in the definition of homogeneity in order to adhere the rule that 0 denotes the unwanted and 1 the desired solution.

- **Completeness:** The completeness criteria describes an opposite situation: a high completeness value indicates that all data vectors that are elements of a given class are also elements of the same cluster. The completeness of a clustering is defined by

$$\begin{aligned} H(C|Y) &= -\sum_{c=1}^{m}\sum_{y=1}^{n}\frac{a_{y,c}}{N}\log\frac{a_{y,c}}{\sum_{i=1}^{n}a_{y,i}} &(3.37)\\ H(C) &= -\sum_{c=1}^{m}\frac{\sum_{y=1}^{n}a_{y,c}}{n}\log\frac{\sum_{y=1}^{n}a_{y,c}}{n}. &(3.38)\end{aligned}$$

  Given these, the completeness c is defined as

$$c = \begin{cases} 1 & \text{if } H(C|Y) = 0, \\ 1 - \frac{H(C|Y)}{H(C)} & \text{else.} \end{cases} \tag{3.39}$$

- **V-measure:** In general, we are interested in clustering solutions that reach both, good completeness and good homogeneity values. Both measures quantify desirable properties, and, usually, a clustering is preferred if the clusters contain only data examples from one class, but also if all data examples that are elements of a given class are elements of the same cluster. The V-measure provides a clustering evaluation measure that incorporates completeness and homogeneity to one value. Therefore, the V-measure V_β is calculated by computing the (weighted) harmonic mean of homogeneity and completeness:

$$V_\beta = \frac{(1+\beta)\cdot h\cdot c}{(\beta\cdot h)+c}.$$

  We can control the influence of completeness and homogeneity by selecting β values according to a particular application. $\beta > 1$ favors good homogeneity, while $\beta < 1$ yield good completeness values.

3.7. Summary

The aim of this chapter was to present algorithms for machine learning of time series as well as their application to interaction time series data. We limited the selection to algorithms that will be used in experiments later in this thesis (Chapter 6 ff).

In **Section 3.1**, we shortly introduced machine learning and feature extraction with a specific focus on approaches to process time series data in **Section 3.2**. We addressed Dynamic Time Warping (Section 3.2.1), Hidden Markov Models (Section 3.2.2), and introduced Ordered Means Models (Section 3.2.3). Please note that Ordered Means Models will be described in detail in the following Chapter 4. In **Section 3.3**, we presented algorithms for classification and their application to time series data. In details, these were Bayes classifier (Section 3.3.1), Nearest Neighbor classifiers (Section 3.3.2), and Support Vector Machines (Section 3.3.3).

Section 3.4 was related to clustering of time series data, and we introduced approaches based on $\mathcal{K}$-means as well as $\mathcal{K}$-trees for hierarchical clustering.

In **Section 3.5**, we investigated how algorithms for machine learning of time series are used to process time series data in interaction scenarios in related work. Here, we focused on recognition, reproduction, and organization of speech, handwriting, and gestures as widely used interaction modalities.

Last, we presented performance measures for classification and clustering in **Section 3.6**. For classification, we first introduced the concept of confusion matrices and related performance measures (Section 3.6.1). Subsequently, we described Receiver Operating Characteristic and the corresponding Area Under Curve measure in Section 3.6.3. For clustering (Section 3.6.4), we focused on extrinsic performance measures that incorporate ground truth, i.e., knowledge about correct clustering solutions. Here, we focused on purity, homogeneity, completeness, as well as V-measure to assess a learned clustering. We will use these measures to evaluate our methods in comparison to other algorithms in experiments, later in this thesis (Chapter 6 ff).

4. Theory and Definition of Ordered Means Models

In this thesis, we use Ordered Means Models (OMMs) for learning of interaction time series. OMMs can be described as a rigorous simplification of HMMs in terms of an exclusion of any transition probabilities. OMMs can still be trained by simple maximum likelihood estimation using the EM algorithm. In this chapter, we describe the theoretical framework of OMMs.

4.1. Model Design

An Ordered Means Model (OMM) is a generative, discrete state space model that emits a time series out of an adjustable number K of model states. The overall model design of OMMs is similar to the model design of HMMs, but instead of explicitly modeling transition probabilities, in OMMs all paths through a network of model states are equally likely. In addition, only transitions to states with equal or higher indexes as compared to the current state are allowed, i.e., the network of model states and state transitions follows a left-to-right topology. See Figure 4.1a for a diagram that illustrates the components and topology of an OMM. Let $O = \mathbf{o}_1, \ldots, \mathbf{o}_T$ be a (multivariate) time series of observation vectors with $\mathbf{o}_t \in \mathbb{R}^d$. Given the above model structure, an OMM is defined by a set of emission densities $B = \{b_k(\mathbf{o_t})\}$, each of which is associated with a particular state k. To model the emission densities we use Gaussian densities

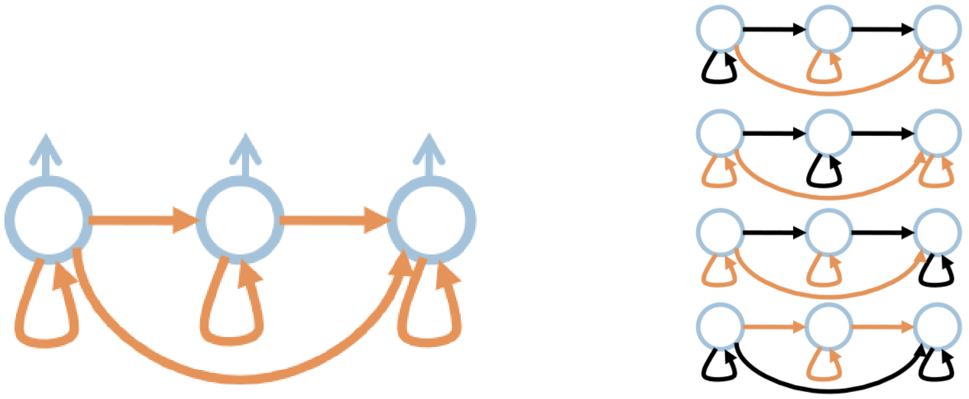
$$b_k(\mathbf{o_t}) = \mathcal{N}(\mathbf{o}_t; \boldsymbol{\mu}_k, \sigma) = \frac{1}{(2 \cdot \pi \cdot \sigma^2)^{d/2}} \cdot \exp \frac{-1}{2 \cdot \sigma^2} ||\mathbf{o}_t - \boldsymbol{\mu}_k||^2. \tag{4.1}$$

The deviation parameter σ describes the deviation of the associated data $\mathbf{o}_t$ from the location parameter $\boldsymbol{\mu}_k$. A low σ value indicates that the data vectors are close to the location parameter, a high σ value denotes wider distributed data examples. In OMMs, the deviation parameter is modeled as a *global hyperparameter*, which implies that the σ value is identical in all states and has to be chosen by an external evaluation or domain knowledge. By varying the deviation parameter σ, it is possible to change the influence of distant data vectors.

The global standard deviation and the left-to-right model topology give rise to models that are mainly defined by a linear array of reference vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, i.e. the location parameters of the emission densities.

For a given length T, we define each valid path $\mathbf{q}_T = q_1, \ldots, q_T$ through the model to be equally likely:

$$P(\mathbf{q}_T|\Omega) = \begin{cases} \frac{1}{M_T} \cdot P(T) & \text{if } q_1 \leq q_2 \leq \cdots \leq q_T, \\ 0 & \text{else} \end{cases} \tag{4.2}$$



(a) This figure illustrates the design of an Ordered Means Model with three states.

(b) Here, all equally likely paths through the model are illustrated by black arrows. Please note that self-transitions can occur multiple times.

Figure 4.1.: OMM with three states and possible paths through the model. The circles picture the model states, the blue arrows represent the emissions from the state, and orange arrows emphasize the allowed state transitions.

where M_T is the number of valid paths for a time series of length T through a K-state model:

$$M_T = |\{\mathbf{q}_T : q_1 \leq q_2 \leq \cdots \leq q_T\}| = \binom{K+T-1}{T}. \tag{4.3}$$

The number of paths through the model is limited by the left-to-right topology of the model. In enumerative combinatorics — an area of mathematics that is related to the studies of finite discrete structures — such a setting is called k-combination with repetitions. The term $\binom{K+T-1}{T}$ denotes the number of ways to sample T elements from a set of K elements and allowing duplicates. Since the order is not taken into account, the elements can be sorted increasingly.

The likelihood to observe the time series O for a given path $\mathbf{q}_T$ and a model Ω then is

$$p(O|\mathbf{q}_T, \Omega) = \prod_{t=1}^{T} p(\mathbf{o_t}|q_t, \Omega) \cdot P(T) = \prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}) \cdot P(T), \tag{4.4}$$

where $P(T)$ denotes the probability to observe a sequence of length T. The likelihood to observe the time series O and a path $\mathbf{q}_T$ for a given model Ω is

$$p(O, \mathbf{q}_T|\Omega) = p(O|\mathbf{q}_T, \Omega) \cdot P(\mathbf{q}_T|\Omega) = \frac{1}{M_T} \prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}) \cdot P(T) \tag{4.5}$$

and the overall production likelihood for a time series of length T corresponds to the sum of all possible paths (denoted as Q) in Equation 4.5

$$p(O|\Omega) = \sum_{\mathbf{q}_T} p(O, \mathbf{q}_T|\Omega) = \frac{1}{M_T} \sum_{q \in Q} \prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}) \cdot P(T). \tag{4.6}$$

4.1.1. Sequence Length Distribution in OMMs

In principle, OMMs require an explicit modeling of the length variable T. Therefore, we have to choose a particular $P(T)$, either by domain knowledge or by estimation from the observed lengths in the training data. However, this may not be possible due to missing knowledge or a biased length of the observations. To circumvent the definition and estimation of a length model we assume a flat distribution in terms of an improper prior according to equally probable lengths. Note that since we do not define a maximum sequence length, the particular probability is unknown and can be assumed to be constant.

4.1.2. State Duration Probabilities in OMMs

State duration probabilities of HMMs depend on the transition probabilities and are geometrically distributed. In OMMs, the probability $P_k(\tau)$ to stay τ time steps in state k depends on the sequence length T and the number of model states K. Considering the combinatorics of the path generation process (see Equation 4.2 and Equation 4.3), the duration probability distributions of OMMs follow

$$P_k(\tau; T, K) = \frac{\binom{T+K-2-\tau}{K-2}}{\binom{T+K-1}{K-1}}. \tag{4.7}$$

According to this distribution, the expected value of the state duration is

$$E[X] = \sum_{\tau=0}^{\infty} \tau \cdot P_k(\tau) = \frac{T}{K}, \tag{4.8}$$

where X is a random variable associated with $P_k(\tau)$.

4.2. Parameter Estimation

In order to estimate the parameters of an OMM, we use a realization of the expectation maximization algorithm.

To learn the model parameters $\boldsymbol{\mu}_k$ from a set of N example time series $\mathbf{O} = \{O^1, \ldots, O^N\}$, we maximize the log-likelihood

$$\mathcal{L} = \ln \left(\prod_{n=1}^{N} p(O^i|\Omega) \right) \tag{4.9}$$

with respect to the mean vectors $\boldsymbol{\mu}_k$:

$$\begin{aligned} \max_{\Omega} \mathcal{L} &= \max_{\Omega} \sum_{i=1}^{N} \ln p(O^i|\Omega) && (4.10) \\ &= \max_{\Omega} \sum_{i=1}^{N} \ln \sum_{q \in Q} p(O^i|\mathbf{q}, \Omega) \cdot P(\mathbf{q}|\Omega) && (4.11) \\ &= \max_{\Omega} \sum_{i=1}^{N} \ln \left(\frac{1}{M_T^i} \sum_{q \in Q} \prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}) \right). && (4.12) \end{aligned}$$

Since this maximization cannot be solved directly, the optimization problem has to be solved iteratively. We use an expectation maximization approach [cf. Dempster et al., 1977] that is similar to Baum-Welch training of HMMs.

This implies a training scheme where we first compute the *responsibilities*

$$r^i_{k,t} \quad = \quad \frac{p(O^i, q_t = k|\Omega)}{p(O^i|\Omega)} \qquad \text{(E-step)} \tag{4.13}$$

and then re-estimate the model parameters according to

$$\boldsymbol{\mu}_k \quad = \quad \frac{\sum\limits_{i=1}^{N}\sum\limits_{t=1}^{T} r^i_{k,t} \cdot \mathbf{o}^i_t}{\sum\limits_{i=1}^{N}\sum\limits_{t=1}^{T} r^i_{k,t}} \qquad \text{(M-step).} \tag{4.14}$$

These two steps are repeated until convergence. Please see Appendix A for details of the expectation maximization algorithm.

4.2.1. Efficient Computation of Production Likelihoods and Responsibilities

To compute the production likelihood (Equation 4.6) and the responsibilities (Equation 4.13) in a computationally efficient way, we use a dynamic program similar to the forward-backward algorithm known from HMMs [Rabiner, 1989]. We define the forward variable according to

$$\alpha_{k,t} = \alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) + \alpha_{k-1,t} \propto p(\mathbf{o}_1, \ldots, \mathbf{o}_t | q_t \leq k, \Omega). \tag{4.15}$$

Here the current $\alpha_{k,t}$ depends only on the variable of the previous state $k-1$ and of the previous point in time $t-1$. This yields an elegant and fast dynamic programming solution:

$$\begin{aligned} \alpha_{k,t} &= \alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) + \alpha_{k-1,t}, & (4.16)\\ \alpha_{k,0} &= 1, & (4.17)\\ \alpha_{0,t} &= 0. & (4.18) \end{aligned}$$

In a similar way we can compute the backward variable

$$\begin{aligned} \beta_{k,t} &= \beta_{k,t+1} \cdot b_k(\mathbf{o}_t) + \beta_{k+1,t} \propto p(\mathbf{o}_t, \ldots, \mathbf{o}_T | q_t \geq k, \Omega), & (4.19)\\ \beta_{k,T+1} &= 1, & (4.20)\\ \beta_{K+1,t} &= 0 & (4.21) \end{aligned}$$

by means of recursion. Note that the terminal values for the forward and backward variables are equal and proportional to the production likelihood:

$$\alpha_{K,T} \quad = \quad \beta_{1,1} \quad \propto \quad p(O|\Omega). \tag{4.22}$$

By combination of Equation 4.15 and 4.19 the responsibilities can be computed by

$$r_{k,t} = \frac{\alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) \cdot \beta_{k,t+1}}{\alpha_{K,T}}. \tag{4.23}$$

4.2.2. Influence of the Standard Deviation Parameter

In our implementation of OMMs the standard deviation σ as well as the model size K are global hyperparameters and must be validated, e.g., by means of $\mathcal{N}$-fold cross validation (cf. Section 5.2.1). In this process, σ controls the influence of distant vectors $\mathbf{o}_t$ on the reference vectors $\boldsymbol{\mu}_k$.

The limiting case $\sigma \to 0$ leads to an expectation maximization procedure with deterministic assignments $z_{k,t}$ of the T-length time series O to the model Ω with $z_{k,t} \in \{0,1\}$ and $\sum_{k=1}^{K} z_{k,t} = 1$. Given this, we only consider the best path $\mathbf{q}_T^*$ through the model resulting in

$$p^*(O|\Omega) = p(O, \mathbf{q}_T^*|\Omega) = \max_{\mathbf{q}_T} p(O, \mathbf{q}_T|\Omega). \tag{4.24}$$

The likelihood that an observed time series is generated by the optimal path can be efficiently computed by the Viterbi algorithm [cf. Lou, 1995]

$$\begin{aligned} v_1(k) &= b_k(\mathbf{o}_1), \\ v_t(k) &= \max_{k'} v_{t-1}(k') \cdot b_k(\mathbf{o}_t) \text{ with } k' < k, \\ p^*(O|\Omega) &= \max_{k} v_T(k) \end{aligned}$$

where $v_t(k)$ is the so-called Viterbi variable [cf. Forney, 1973]. Combined with a *back tracking matrix* Φ, we can find the optimal path and the assignments $z_{k,t}$.

Since each path is equally likely $(p(\mathbf{q}|\frac{1}{M}))$ is the probability to observe O for a given OMM Ω under the optimal path $\mathbf{q}^*$:

$$\begin{aligned} p^*(O|\Omega) &= P(\mathbf{q}^*|\Omega) \cdot \ p(O|\mathbf{q}^*, \Omega) && (4.25) \\ &= \frac{1}{M} \prod_{t=1}^{T} b_{q_t^*}(\mathbf{o}_t). && (4.26) \end{aligned}$$

If this is inserted into the objective function (Equation 4.10), we get:

$$\max_{\Omega} \mathcal{L} = \max \sum_{i=1}^{N} \ln p^*(O^i|\Omega) = \min \sum_{i=1}^{N} \sum_{t=1}^{T^i} \sum_{k=1}^{K} z_{k,t}^i ||\mathbf{o}_t^i - \boldsymbol{\mu}_k||^2 \tag{4.27}$$

where the $z_{k-t}^i \in \{0,1\}$ denote the assignments of $\mathbf{o}_t^i$ to $\boldsymbol{\mu}_k$.

This can be solved according to a re-estimation scheme (cf. Equation 6.8) by

$$\boldsymbol{\mu}_k = \frac{\sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T} z_{k,t}^i \cdot \mathbf{o}_t}{\sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T} z_{k,t}^i}. \tag{4.28}$$

4.3. Numerical Aspects

The implementation of the dynamic programming as described in Section 4.2.1 requires considerations about numerical aspects.

Consider the definition of

$$\alpha_{K,T} \quad \propto \quad p(O|\Omega) \quad = \quad \frac{1}{M_T}\sum_{q\in Q}\prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}). \tag{4.29}$$

Here $b_{q_t}(\mathbf{o_t})$ is, since it is a probability, in the numerical range between 0 and 1. For large T, $\prod_{t=1}^{T} b_{q_t}(\mathbf{o_t})$ heads exponentially towards zero and will quickly exceed the numerical precision range of today's computer. E.g., if we assume all $b_{q_t}(\mathbf{o_t}) = 0.1$, and choose a time series length of $T = 100$, $\prod_{t=1}^{T} b_{q_t}(\mathbf{o_t}) = 10^{-100}$ — a number that cannot be represented properly in terms of floating point precision.

To avoid numerical instabilities, we developed two approaches:

1. Computation in log space,
2. Incorporation of a scaling procedure.

4.3.1. Computation in Log Space

In order to transform probabilities to a suited numerical range, a common approach to multiplication of probabilities is to sum their logarithms. In case of OMMs, the emission densities are then represented by distances

$$\ln \mathcal{N}(\mathbf{o}_t; \boldsymbol{\mu}_k, \sigma) \approx \frac{-1}{2\cdot\sigma^2}\cdot ||\mathbf{o}_t - \boldsymbol{\mu}_k||^2. \tag{4.30}$$

In log space, products are substituted by sums, and the Kingsbury-Rayner formula allows fast summation

$$\log(a+b) = \log(a) + \log(1 + b/a) = \log(a) + \log(1 + \exp(\log(b) - \log(a))). \tag{4.31}$$

In addition, many programming languages provide fast function calls for computation of $\log(1+x)$ (e.g. the Python or C programming languages).

4.3.2. Scaling of Probabilities

Another approach to multiply many probabilities is to incorporate a scaling procedure.

In case of OMMs, the goal is to scale all $\alpha_{k,t}$ and $\beta_{k,t}$ to a dynamic range that is feasible for a computer. This should be enabled in a way that in the end, i.e., for computation of the responsibilities $r_{k,t}$ and the production probabilities $p(\mathbf{o}_t|\Omega)$, all scaling factors are canceled out.

Consider the scaling coefficient

$$c_t = \frac{1}{\sum_{k=1}^{K}\alpha_{k,t}}. \tag{4.32}$$

We then can write by induction

$$\hat{\alpha}_{k,t} = c_t \cdot \hat{\alpha}_{k,t} = \left(\prod_{u=0}^{T} c_u\right)\cdot \alpha_{k,t} \tag{4.33}$$

according to the dynamic programming in Equation 4.16. Scaling $\beta_{k,t}$ in a similar manner

$$\hat{\beta}_{k,t} = c_t \cdot \beta_{k,t} \tag{4.34}$$

yields a re-formulation of the responsibilities (Equation 4.23) by

$$\begin{aligned} r_{k,t} &= \frac{\alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) \cdot \beta_{k,t+1}}{\alpha_{K,T}} && (4.35)\\ &= \frac{\left(\prod_{u=0}^{T} c_u\right) \cdot \alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) \cdot \beta_{k,t+1}}{\left(\prod_{u=0}^{T} c_u\right) \cdot \alpha_{K,T}}. && (4.36)\\ &= \frac{\left(\prod_{u=0}^{t-1} c_u\right) \cdot \alpha_{k,t-1} \cdot b_k(\mathbf{o}_t) \cdot \left(\prod_{u=t}^{T} c_u\right) \cdot \beta_{k,t+1}}{\left(\prod_{u=0}^{T} c_u\right) \cdot \alpha_{K,T}}. && (4.37) \end{aligned}$$

The scaling factor maps the magnitude of α_t and β_t to 1 and, thus, in a numerical reasonable range.

Similarly, the production likelihood then is

$$p(O|\Omega) = \frac{1}{M}\alpha_{T,K} = \frac{1}{M \cdot \left(\prod_{u=0}^{T} c_u\right)}\alpha_{K,T}. \tag{4.38}$$

4.4. Relation to Hidden Markov Models

At first sight, the general design of OMMs is similar to other graphical probabilistic models, most notably HMMs. However, there are important differences that make OMMs particularly suitable for time series of the kind we want to deal with here. OMMs are defined without any state transition probabilities, which leads to fundamentally modified state duration probabilities. In OMMs, the probability $P_k(\tau)$ to stay τ time steps in state k depends on the time series length T and the number of model states K and, thus, varies for time series of different length. Considering the combinatorics of the path generation process (see Equation 4.2 and Equation 4.3), the duration probability distributions of OMMs follow

$$P_k(\tau; K, T) = \frac{\binom{T+K-2-\tau}{K-2}}{\binom{T+K-1}{K-1}}. \tag{4.39}$$

In contrast, the state duration probabilities of HMMs depend on the transition probabilities and are geometrically distributed. There is no equivalent realization of such a modeling in terms of transition probabilities in HMMs. In addition, the time complexity of OMMs is $O(K \cdot T)$, i.e., linear in the number of model states K and the length of a time series T. The time complexity of HMMs that incorporate a similar left-to-right topology is, with $O(K^2 \cdot T)$, polynomial in the number of model states K. In practical applications, this can make an enormous difference. Consider the quite realistic case of models with $K = 100$ states and a time series of $T = 100$ length (i.e., about 3 seconds duration with a common sampling rate of 30 Hz). With c being a constant factor, OMMs would need $c \cdot 10^4$ steps of calculation, a similar HMM $c \cdot 10^6$ computational steps. In interaction scenarios where fast response times are crucial, the reduced complexity of OMMs can improve the reaction times considerably.

4.5. Summary

In this chapter, we introduced Ordered Means Models (OMMs), a probabilistic discrete state-space model that is the base of this thesis' algorithmic work. The general model design of OMMs is similar to the model design of HMMs, but instead of explicitly modeling transition probabilities, in OMMs all paths through a network of model states are equally likely. In addition, only transitions to states with equal or higher indexes as compared to the current state are allowed, and the emission distribution are modeled as Gaussians with the standard deviation parameter used as a global hyperparameter.

In **Section 4.1**, we presented the general model design of OMMs. Here, we were able to show that OMMs' state duration probabilities depend on the number of model states and the length of the observed time series. In **Section 4.2**, we described iterative training schemes to estimate appropriate model parameters for a set of observation by a realization of the Expectation Maximization algorithm. Here we demonstrated that, despite a flexible left-to-right model topology, the computational complexity of OMMs' training is linear in the number of model states and the average time series length. In addition, we considered the limiting case of the deviation parameter approaching 0, what leads to an alternative training scheme where only the optimal path through the model states is considered. After considering difficulties in representing probabilities as floating point numbers in **Section 4.3**, we subsequently presented two solutions to avoid numerical instabilities: computation in log-space and scaling the probabilities to a magnitude of 1. Last, we discussed the relation of OMMs and HMMs and were able to demonstrate that the state duration distribution of OMMs cannot be realized by means of transition probabilities in HMMs (**Section 4.4**).

5. Experiments: Methods, Data Sets, and Application

In order to evaluate if Ordered Means Models are able to represent interaction time series data, we compare our approach to a variety of standard methods for machine learning of time series data. We evaluate these methods with various data sets from different data domains. This chapter introduces the standard machine learning algorithms that we use for comparison (Section 3.6), the evaluation schemes we choose to address this work's research questions (Section 5.5), as well as an overview of the data sets (Section 5.6).

5.1. Methods

To determine the abilities of OMM-based methods for recognition, reproduction, and organization of time series data, we compare them to standard approaches to time series analysis from classification, clustering and prototyping of time series.

5.1.1. Recognition

For recognition of interaction time series, we use standard algorithms for classification of time series. On one hand, we compare OMMs to memory-based approaches such as Nearest-Neighbor classifiers and Support Vector Machines. On the other hand, we use generative, probabilistic models, i.e., various variants of Hidden Markov Models. In detail, the following time series classifiers are used in experiments:

- $\mathbf{NN_{DTW}}$**:** As a base-line approach, we use Nearest Neighbor classifiers that incorporate Dynamic Time Warping as a distance function. The time complexity of $\mathrm{NN}_{\mathrm{DTW}}$ is $O(N \cdot T \cdot \bar{T} \cdot d)$ where N denotes the number of training examples, T is the length of the time series that is to be classified, $\bar{T}$ is the average length of the labeled training data, and d refers to the data dimensionality. A Nearest Neighbor classifier does not require training nor involve hyperparameter selection.
- $\mathbf{NN_{DTW\text{-}window}}$**:** This Nearest Neighbor classifier also uses Dynamic Time Warping as a distance function. In contrast to $\mathrm{NN}_{\mathrm{DTW}}$, $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$ comprises the Sako-Chiba band in the DTW function. The algorithmic complexity of $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$ is still $O(N \cdot T \cdot \bar{T} \cdot d)$, but pruning the DTW search space by means of a Sako-Chiba band significantly reduces the complexity in practice, and in an optimal case, the time complexity of $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$ can be linear in all factors: $O(N \cdot \bar{T} \cdot d)$. In contrast to a standard $\mathrm{NN}_{\mathrm{DTW}}$ approach, $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$ requires a thorough selection of the Soka-Chiba band window size, e.g., by means of an external process such as $\mathcal{N}$-fold cross validation (cf. Section 5.2). In our implementation of $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$, the window size parameter is mapped to $[0, 1]$ with 0 denoting a window size of one

sample (i.e., a linear solution that limits the number of possible DTW paths to one), while 1 is related to a full DTW paths search.

- **SVM$_{\mathbf{GDTW}}$:** More sophisticated approaches to classification are provided by Support Vector Machines with Gaussian Dynamic Time Warping kernels [SVM$_{\text{GDTW}}$, Bahlmann et al., 2002, cf. Section 3.2.1]. A Gaussian Dynamic Time Warping kernel is a standard RBF kernel that incorporates a Dynamic Time Warping distance instead of the Euclidean distance into the exponent. The time complexity of SVM$_{\text{GDTW}}$ for classification is with $O(N_{SV} \cdot T \cdot \bar{T} \cdot d)$ equivalent to NN$_{\text{DTW}}$. However, SVM$_{\text{GDTW}}$ only embraces N_{SV} support vectors in a classifier. Please note that in dependency of a given data set, the amount of support vectors N_{SV} might be substantially smaller than the number of all data example N.

  During training, SVM classifiers require to compute pairwise kernel functions for the complete training data set. Thus, the time complexity of SVM$_{\text{GDTW}}$ is polynomial in the number of training data $O(N^2 \cdot \bar{T}^2 \cdot d)$. In addition, SVM$_{\text{GDTW}}$ requires two hyperparameters: the kernel function parameter λ, and the regularization parameter C. These parameters have to be selected, usually, by an external hyperparameter selection process such as $\mathcal{N}$-fold cross validation. As SVM software, we use libsvm [Chang and Lin, 2011] with pre-computed kernel matrices in experiments.

In addition to those standard approaches to time series classification, we use different HMM variants that, just as OMMs, incorporate a simplification in model design. In general, OMMs provide a model with reduced model complexity. In this thesis, we use HMM variants with Gaussian emission densities and use the variation parameter as a hyperparameter. In addition, we further reduce the model complexity by limiting the model topology in regard to (a) the number of transition probabilities, and (b) the requirement to estimate the transition probabilities at all:

- **HMM$_{\mathbf{LR}}$:** HMM$_{\text{LR}}$ are Hidden Markov Models that incorporate the same model topology as OMMs, i.e., a left-to-right model topology that only allows transition to states with equal or higher indexes than the current state. Similar to OMMs, the emissions of HMM$_{\text{LR}}$ are Gaussian. The time complexity of HMM$_{\text{LR}}$ is with $O(K^2 \cdot T)$ polynomial in the number of model states K and linear in the length T of the analyzed time series.
- **HMM$_{\mathbf{LIN}}$:** This HMM variant incorporates a similar design as HMM$_{\text{LR}}$. However, instead of comprising a left-to-right model topology, HMM$_{\text{LIN}}$ incorporate a linear topology where only self-transitions or transitions to the next state are allowed. The time complexity of HMMs$_{\text{LIN}}$ is, similar the complexity of OMMs, linear in the number of model states K and the lengths T of the analyzed time series $O(K \cdot T)$. HMMs$_{\text{LIN}}$ do not provide such a flexible model topology such as HMM$_{\text{LR}}$ or OMMs.
- **HMM$_{\mathbf{PDTP}}$:** HMM$_{\text{PDTP}}$ are an HMM variant with pre-defined transition probabilities. The model complexity of HMMs is decreased by eliminating the requirement to estimate transition probabilities during training. There are various ways to do so, e.g., naïve approaches with uniform transition probabilities or models with randomly chosen transition probabilities. In HMM$_{\text{PDTP}}$, we apply prior knowledge to define the transition probabilities.

  In general, the probability to stay τ time steps in the k-th state of an HMM is geometrically distributed with

$$P_k(\tau) = a_{k,k}^{\tau-1} \cdot (1 - a_{k,k}). \tag{5.1}$$

For a model with K states and a time series of length T, we can require each state to emit $\frac{T}{K}$ time series positions on average. Given this, the expectation value of the geometric distributed random variable X with $P(X) = P_k(\tau)$ is

$$E[X] = \frac{1}{(1 - a_{k,k})} = \frac{T}{K}. \tag{5.2}$$

By solving this equation, we can compute the self-transition probabilities for all states via

$$a_{k,k} = \frac{T - K}{T}, \text{ with } T \neq 0. \tag{5.3}$$

In models with linear topology, where only self transitions $a_{k,k}$ and transitions to the subsequent state $a_{k,k+1}$ are allowed, the probability to change to the next state then is

$$a_{k,k+1} = 1 - a_{k,k} = \frac{K}{T}. \tag{5.4}$$

The transition probabilities are completely determined by such an approach. In combination with Gaussian emission densities and a fixed global standard deviation, this approach leads to reduced HMMs, for which only the location parameters of the emissions have to be estimated. The general process of parameter estimation is similar to standard HMMs.

5.1.2. Organization

In order to evaluate the performance of the proposed OMM-based clustering algorithms, we use standard clustering of time series data.

- **$\mathcal{K}$-means clustering:** In case observations of equal lengths are available, we use standard $\mathcal{K}$-means clustering for comparison (cf. Section 3.4.1 for a detailed description). Please note that such an approach is highly artificial since time series data usually differ in example lengths. The algorithmic complexity of $\mathcal{K}$-means is $O(C \cdot N \cdot d)$, i.e., linear in all factors: the number of observations N, the number of centroids C, and the data dimensionality d.
- **$\mathcal{K}$-HMMs clustering:** This approach applies HMMs to represent the cluster centroids, thus, allowing the observations to be time series data of varying lengths. To apply HMMs to clustering analysis, an available distance function between a time series O^i and an HMM λ^j is defined via the negative production log-likelihood

$$d(O^i, \lambda^j) = -\ln p(O^i|\lambda^j). \tag{5.5}$$

Thus, longer distances are related to lower production likelihood values, while short distances correspond to high production likelihoods. The update function for the centroids then is a re-training of the centroid HMMs. In this thesis, we use HMMs with a similar model topology as OMMs, i.e., a flexible left-to-right topology, and emissions that are modeled as Gaussians whereas the standard deviation parameter is a global hyperparameter. The algorithmic complexity of the $\mathcal{K}$-HMMs algorithm then is $O(C \cdot N \cdot T \cdot K^2)$, i.e., linear in the number C of centroid HMMs, the number of observations N, the length of the observations T, and polynomial in the number of model states K. The $\mathcal{K}$-HMMs algorithm for clustering of time series was first introduced by Smyth [1997].

5.1.3. Prototype Property

Similar to OMMs, the emission densities of HMM_{LR}, HMM_{LIN}, and HMM_{PDTP} are Gaussians and the standard deviation parameters are used as global hyperparameters. Thus, the expectation values of the emission densities can be described as a linear array of reference vectors. Moreover, since these reference vectors are elements of the same vector space as the observations from the genuine time series space, the complete array of reference vectors are elements of the time series data, too. However, given the model design of HMMs transition probabilities are necessary to completely describe HMMs_{LR} and HMMs_{LIN} — an information that is unused in such a prototype definition.

5.2. Hyperparameter Selection

Most machine learning algorithms require a configuration of hyperparameters. In case of OMMs, the hyperparameters are the number of model states K and the standard deviation parameter σ, for standard SVMs, the penalty weight C and kernel parameters, e.g., the kernel bandwidth parameter γ have to be determined. In general, the choice of appropriate hyperparameters is crucial for the algorithm's performance. In order to choose a particular hyperparameter set, we use $\mathcal{N}$-fold cross validation in this thesis.

5.2.1. $\mathcal{N}$-Fold Cross Validation

Cross validation is a common approach to select hyperparameter and to evaluate a classifier's performance. The general idea of cross validation is to hold a subset of a complete training data set back and use it for testing.

In order to apply cross validation to a data set, the data set is partitioned into $\mathcal{N}$ folds of (almost) equal size. One of these folds is taken out for testing, while the remaining $\mathcal{N} - 1$ folds are merged and used for training. This procedure is repeated for all $\mathcal{N}$ folds and an average of all $\mathcal{N}$ folds' performances is used to rate the parameter set or the learner, respectively.

In a more formal definition is $\kappa : \{1, \dots, N\} \to \{1, \dots, \mathcal{N}\}$ a function that randomly maps an observation $i \in \{1, \dots, N\}$ in a fold $g \in \{1, \dots, \mathcal{N}\}$. And let $f^{-\kappa(g)}(\mathbf{x})$ be a model that was trained without the data from the g's fold. The estimated error from the cross validation procedure is

$$\hat{E}_{CV} = \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \sum_{j=1}^{N^{-\kappa(i)}} L(f^{-\kappa(i)}(\mathbf{x}^j), y^j).$$

Common values for the number of folds $\mathcal{N}$ are 5 or 10.

5.3. Initialization of OMMs and HMMs

For initialization of the iterative model estimation scheme, we choose randomized deterministic assignments to model states for all time series. The resulting random paths are forced to follow the restrictions induced by the model topologies. The initial transition probabilities for HMMs are chosen according to (normalized) uniform random numbers. All algorithms used in this study are implemented in the Python programming language. Time-critical parts such as the dynamic programming are realized using the C programming language. The complete source code is available on the complementary CD.

The experiments were performed on a computing cluster with 16 nodes and 128 processors.

5.4. Data Pre-Processing

All data is normalized to zero-mean and unit variance. Let $\mathbf{O} = \{O^1, \ldots O^N\}$ be a set of observed time series $O^n = \boldsymbol{o}_1^n \ldots \boldsymbol{o}_T^n$. In order to normalize the data, we first estimate the mean vector for all observations

$$\hat{\boldsymbol{\mu}} = \frac{1}{N \cdot \sum_{n=1}^{N} T_n} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \boldsymbol{o}_t^n,$$

followed by an estimation of the data's standard deviation

$$\hat{\sigma} = \frac{1}{N \cdot \sum_{n=1}^{N} T_n} \left(\sum_{n=1}^{N} \sum_{t=1}^{T_n} (\boldsymbol{o}_t^n - \hat{\boldsymbol{\mu}})^2 \right)^{\frac{1}{2}}.$$

The normalized data sample is then computed by

$$\tilde{\boldsymbol{o}}_t = \frac{\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}}{\hat{\sigma}}.$$

5.5. Application in Experiments

To establish the classifiers' comparative merits, we apply a uniform procedure to all data sets and models in this work: first, we identified optimal hyperparameter values for classifiers — unless otherwise stated — by means of 5-fold cross validation on the training data. Subsequently, we train a classifier using all training data and the hyperparameters associated with the highest performance measure value. To obtain the final performance rate, we apply the resulting classifiers to the dedicated test data set.

We evaluate the reproduction capabilities of our approaches by visual inspection. E.g., mouse gestures can be easily inspected in terms of 2-dimensional spatial diagrams.

In order to evaluate the performance of the organization algorithms, we apply uniform procedures to all data sets and methods. We employ the organization methods with different hyperparameter value sets to a given data set, and evaluate the performance by means of appropriate performance measures for clustering (cf. Section 3.6.4).

In other experiments, we evaluate the organization capabilities of an algorithm by means of test set performance, similar to the recognition experiments. Here, we estimate appropriate hyperparameters for each method by means of the training data. Subsequently, we take the best hyperparameters found in this process to train a model from the complete training data sets and apply the resulting clustering model to the test data sets.

Unless otherwise stated, we initialize the time series clusters with randomized cluster assignments.

5.6. Overview of Data Sets

We use a variety of data sets to evaluate and demonstrate the properties of OMMs — in particular related to time series that arise in interaction scenarios, but also in relation to time series analysis in general and from other domains. Alongside to widely used benchmark data sets for time series classification and clustering provided by Keogh et al. [2006], we use interaction time series data sets from the UCI machine learning repository, as well as data sets that are genuine to this thesis. Table 5.1 gives an overview of basic properties for the used data sets, while Table 5.2 shows what data set is used in which experimental context. In detail the data sets are:

- **The UCR Time Series Classification/Clustering Data Sets:** This collection of time series data sets provides 20 labeled data sets from various data domains represented as time series[1]. The collection includes data sets with various properties and claims to provide "a public service to the data mining/machine learning community, to encourage reproducible research for time series classification and clustering" [Keogh et al., 2006]. A shortcoming of these data sets for the purpose of the given thesis is that all time series from a single data set share an equal length. Such a prerequisite usually requires a pre-processing step in which, e.g., the recorded time series are interpolated to equal lengths. However, in interaction scenarios, such a pre-processing step is, usually, impractical due to time constraints or an incremental evaluation approach. All data sets in this collection are uni-variate.
- **The Australian Sign Language Signs[2]:** This data set is provided by the UCI machine learning repository. The data set contains recordings from sign language signs that were collected from a single right-handed Nintendo Power Glove at a sample rate of approximately 23 Hz. With 95 classes, this data set comprises the largest amount of classes in this thesis. Kadous [1995] recorded this data set for an honors thesis.
- **The Character Trajectories[3]:** The character trajectories data set is also provided by the UCI machine learning repository. This data set contains trajectories of characters that were written on a WACOM tablet by various human participants. A character is represented as a time series of three dimensional feature vectors (x/y coordinates and pen force). We randomly partitioned the data set in training and test sets of equal sizes. The data set was collected by Ben Williams [e.g., Williams et al., 2006] and was primarily used for a PhD study.

[1] Please note, that when the experiments with the UCR data sets where conducted, the collection contained 20 data sets. By now (13.02.2013), the collection has grown to 47 data sets.

[2] http://www.cse.unsw.edu.au/˜waleed/tml/data/

[3] http://archive.ics.uci.edu/ml/datasets/Character+Trajectories

	data set	#classes	#train examples	#test examples	avg. length	#dimensions
UCR benchmark data sets	50words	50	450	455	270	1
	Gun_Point	2	50	150	150	1
	OSULeaf	6	200	242	427	1
	Two_Patterns	4	1000	4000	128	1
	Adiac	37	390	391	176	1
	ECG200	2	100	100	96	1
	wafer	2	1000	6164	152	1
	Beef	5	30	30	470	1
	FaceAll	14	560	1690	131	1
	Lighting2	2	60	61	637	1
	SwedishLeaf	15	500	625	128	1
	yoga	2	300	3000	426	1
	CBF	3	30	900	128	1
	FaceFour	4	24	88	350	1
	Lighting7	7	70	73	319	1
	synthetic_control	6	300	300	60	1
	Coffee	2	28	28	286	1
	fish	7	175	175	463	1
	OliveOil	4	30	30	570	1
	Trace	4	100	100	275	1
UCI interaction data sets	Australian Sign Language Signs	95	1710	570	57	22
	Character Trajectories	20	1429	1429	171	3
	Spoken Arabic Digits	10	6599	2199	39	13
original data sets	Concrete Gestures	48	369	151	141	3
	Abstract Gestures	9	362	147	144	3
	SCH3	5	289	150	21	6
	Head Gestures	4	751	357	137	6
	Mouse Gestures	3	11	-	100	2
	Violin Finger Pressure	3	1143	1143	14	4

Table 5.1.: This table denotes basic properties of the data sets used in the experiments. In detail, these are the number of classes, the number of training examples, the number of test examples, the average length of the time series, and the dimensionality of the time series. The original data sets were recorded in different studies for this thesis.

- **The Spoken Arabic Digits**[4]**:** As both data sets above, the Spoken Arabic Digits data set is available through the UCI machine learning repository. It provides 8800 examples of spoken Arabic digits in 10 classes, each digit in one class. The spoken digits are represented by time series of 13 dimensional mel frequency cepstral coefficients [Logan et al., 2000]. This data set provides dedicated training and test data sets and was recorded by the Laboratory of Automatic and Signals, University of Badji-Mokhtar, Annaba, Algeria.
- **Concrete Gestures:** This data set was originally recorded by Amir Sadeghipour in cooperation with the author of this thesis. We recorded hand-arm gestures as 3-dimensional location coordinates of the right hand wrist that vary in time by means of a time-of-flight camera and a marker-free tracking software. Overall, we collected 48 different gesture classes, each with respect to some of the following variant features: size (e.g., small and big circle), performing space (e.g., drawing a circle at the right side or in front of oneself), direction (clockwise or counter-clockwise), orientation (horizontal or vertical), repetition (repeating some sub-parts of the movement, such as drawing a circle once or twice, or swinging the hand for several times during waving). The performed gestures ranged from conventional communicative gestures ("waving", "come here" and "calm down") over iconic gestures ("circle", "spiky", "square", "surface" and "triangle") to deictic gestures ("pointing" to left, right and upward).
- **Abstract Gestures:** This data set comprises the same gesture trajectories as the before-mentioned Concrete Gestures data set. In contrast to the Concrete Gestures data set, the Abstract Gestures data set combines the trajectories differently to only nine *abstract* gesture classes. These classes comprise gestures from higher-level semantic categories, e.g., all circle gestures are combined to one class "circle" despite of their size, performance space, direction, orientation, or repetition. This leads to classes that enclose very different trajectories in terms of the given raw data, and, thus, are more challenging to represent by a data model.
- **SCH3:** The SCH3 data set contains trajectories of human players who play a modified version of the common Rock-Paper-Scissors hand gestures game against the virtual agent VINCE. In contrast to its classic form, the opponents had to use hand-arm gestures to represent the RPS items. Besides that, we also added two more items to the game (Lizard and Spock) to provide a more comprehensive data collection. In this setup, we captured the gestures that were performed by the human users as normalized 3-dimensional positions of the wrists with respect to the body center. The data set was recorded by Wöhler [2012] as part of his Master's thesis under close supervision of Amir Sadeghipour and the author of this thesis.
- **Mouse Gestures:** The Mouse Gestures data set is a very small data collection that comprises computer mouse trajectories in three classes with 11 examples per class. The mouse gestures are represented as 2D location coordinates that vary in time. This data set was originally recorded by the author of this thesis.
- **Head Gestures:** This data set provides angular momentum data of conversation head gestures. Therefore, we recorded head movements of 10 human participants during dyadic conversation by means of the 3 DoF gyroscopes of a Xsens MT9

[4] http://archive.ics.uci.edu/ml/datasets/Spoken+Arabic+Digit

	Experiments				
data sets	Reproduction	Recognition	Incremental Recognition	Organization	Hierarchical Organization
UCR Time Series Classification/Clustering Data Set		✓		✓	
Australian Sign Language Signs		✓			
Character Trajectories		✓			
Spoken Arabic Digits		✓			
Concrete Gestures	✓	✓			✓
Abstract Gestures	✓	✓			✓
SCH3			✓		
Mouse Gestures	✓				
Head Gestures		✓			
Violin Finger Pressure		✓			

Table 5.2.: This table shows the usage of data set in the experimental contexts. Please note, that the UCR Time Series Classification/Clustering data sets is a collection of 20 benchmark data sets from various fields that is provided by Keogh et al. [2006].

inertial sensor[5] that was mounted to the top of the participants' heads. Afterward, we annotated the recorded data by means of the video footage recorded by a scene camera according to the four most frequently occurring head movements: *nod*, *shake*, *tilt*, and *look*. This data set was collected by Wöhler [2009] as part of his Bachelor's thesis under close supervision of Angelika Dierker in cooperation with the author of this thesis.

- **Violin Finger Pressure:** This data set contains finger pressure measurements of a violin player performing. We recorded the pressure that the fingers of the right hand put on the bow. Therefore, we mounted force sensing resistors to the bow's frog. In addition, we used a goniometer to measure the direction and range as well as the rotation of the bow's motion. The data set comprises different up- and down bowings (Martele, Detache and Spiccato) in different tempi (slow and fast), and defined inaccurate bowings. The data set was recorded by Tobias Grosshauser in cooperation with the author of this thesis.

[5] http://www.xsens.com/

5.7. Summary

In this chapter, we introduced the algorithms that we chose for comparison and the evaluation schemes that we use in experiments. Furthermore, we gave an overview of the data sets that are employed in empirical evaluations.

In **Section 5.1** we presented the standard machine learning algorithms that we use for comparison of our OMM-based recognition, reproduction, and organization approaches. In addition to Support Vector Machines with Gaussian Dynamic Time Series kernel (SVM_{GDTW}) and Nearest Neighbor approaches that incorporate Dynamic Time Warping as a distance function (NN_{DTW}, $\text{NN}_{\text{DTW-window}}$), we introduced Hidden Markov Models with different characteristics: (i) HMMs_{LR} incorporate the same model design as OMMs but incorporate transition probabilities, (ii) HMMs_{LIN} use, in contrast to OMMs and HMMs_{LR}, a linear model topology, and (iii) $\text{HMMs}_{\text{PDTP}}$ — an HMM variant with pre-defined transition probabilities. In addition to evaluation of OMMs recognition capabilities, these HMMs are also used to evaluate the abilities of OMMs to provide prototype representations for interaction data. To establish OMM-based organization merits, we presented $\mathcal{K}$-means as well as $\mathcal{K}$-HMMs that we use for comparison.

Afterward, we described the application of the algorithms in experiments with real-world data, e.g., practical aspects of data pre-processing and model selection (**Section 5.2**). In **Section 5.3**, we described the initialization of OMM's and HMM's iterative training scheme, while **Section 5.4** specified the data normalization. In **Section 5.5**, we described how the methods are applied in experiments. Last, we gave a short and condensed overview of the 28 data sets that are used in this thesis in **Section 5.6**, alongside basic data set properties.

6. Recognition of Interaction Time Series Data with OMMs

Interacting with one another means engaging in a highly dynamic process in which expressive behaviors are flexibly and spontaneously observed, taken up, recognized, adapted, and adopted (cf. Chapter 2). This includes the ability to rapidly and robustly recognize an observed behavior, thus, recognizing interactive behavior in highly dynamical environments confronts technical systems such as autonomous agents or robots with various challenges: (i) In order to allow a seaming-less interaction experience without substantial delays, it is essential that a system can recognize behavior patterns rapidly with low response times. (ii) Coping with data sources in highly dynamic environments implies a high chance for faulty, corrupt data streams, e.g., if a sensor fails, the available data will be incomplete. However, the human participants themselves might induce an even bigger challenge: being accustomed to deal with reliable human interactants, they might perform behavior patterns only loosely or partially, leaving the observer to deduce the meaning by context of knowledge. (iii) In order to provide a seaming-less communication experience without noticeable delay, it is essential, that a system can recognize a behavior and deduce its (possible) meaning incrementally while it is performed. (iv) To adapt to a dynamical environment the system will highly benefit from the ability to learn a behavior already from the first presentation on, and to adjust the learned model incrementally when observing further instances of this behavior.

In the following chapter, we will investigate Ordered Means Models for classification of time series data with a major focus on interaction time series.

Our experimental setups are directly related to the following major research questions of this thesis (cf. Section 2.1):

- **(Q1) Accuracy:** How do OMM-based classifiers compare to standard methods for classification of time series data in terms of accuracy?
- **(Q2) Response Times:** How do OMM-based classifiers compare to standard methods for classification of time series data in terms of response times?
- **(Q3) Robustness Property:** How do OMM-based classifiers perform in terms of incomplete observations in comparison to standard classifiers?
- **(Q4) Adaptability:** To what degree can OMMs dynamically adapt to changing behaviors?
- **(Q5) Incremental Evaluation:** Does an incremental evaluation of OMMs allow an on-the-fly classification of observed behavior, and can this improve the time required for recognition?

The chapter is organized as follows:
Section 6.1 is related to classification of time series and gesture data with OMMs. Here,

we will evaluate if OMM-based classifiers are suited for recognition of time series data and, in particular, interaction time series. We will apply OMM classifiers to simulation and benchmark data sets for time series classification, and we will introduce two applications of OMM classifiers in interaction scenarios: (i) for communicative head gesture recognition, and (ii) for recognition of bowing-types and malpositions in bowing instrument playing. In Section 6.7, we will evaluate an incremental classification approach that updates a classification decision while a behavior is observed. In addition, the model is dynamically updated while the data arrives. Here, we examine an interaction scenario where a human plays a variant of the well-known Rock-Paper-Scissors game against the virtual agent VINCE.

6.1. Classification with Ordered Means Models

In order to recognize time series data, we will use a maximum likelihood classification approach to OMMs.

In this classification paradigm, we first estimate on OMM Ω for each class $y_j \in Y$ from the data. An unknown time series then is assigned to the class associated with the model that yields the highest production likelihood $p(O|\Omega_i)$ of all models:

$$f(O) = \arg\max_{y_i} p(O|\Omega_i) = \arg\max_{y_i} \alpha^{y_i}_{K,T}. \tag{6.1}$$

6.1.1. Incremental Classification of Time Series with OMMs

Incremental classification of time series data describes the process of yielding a classification result while a time series data is observed and, thus, the classification decision is updated each time new information is available, e.g., when a new sample in time arrives. Consider the time series $O = \mathbf{o}_1, \ldots, \mathbf{o}_T$. The classifier $f(\cdot)$ returns a hypothesis $f(\mathbf{o}_1) = \hat{y}_{t=1}$ when the first sample $\mathbf{o}_1$ is observed. However, if new data, i.e., more samples of the time series, becomes available, the hypothesis gets updated and, supposedly, improved: $f(\mathbf{o}_1, \ldots, \mathbf{o}_t) = \hat{y}_{t=t}$. Since the classification decision is updated incrementally, and available at present at any time during observation of the time series, such an approach is an incremental classification of time series, and we will refer to corresponding OMMs by $\text{OMMs}_{\text{incremental}}$.

The dynamic programming scheme of OMMs (cf. Equation 4.16) allows an incremental evaluation of time series. Since the value of $\alpha^i_{K,t}$ only depends on $\alpha^i_{K,t-1}$ — the values of the forward variables that are related to the previous observed sample — it is possible to update the production likelihood $P(o^i_1, \ldots, o^i_t|\Omega_j) \propto \alpha^i_{K,t}$ sample-by-sample, which is the basis for the classification decision. Notably, this update scheme leads to incrementally improved classification decisions with every further piece of evidence that arrives. In consequence, a classification decision can be made with specific required certainties as soon as possible, and the system can wait for and process further evidence otherwise.

In order to enable incremental classification of time series based on OMMs, the dynamic programming approach as described in Equation 4.16 need to be altered. A cumulative computation scheme of the production probabilities is

$$
\begin{aligned}
p(\mathbf{o}_1, \ldots, \mathbf{o}_t|\Omega) &= \frac{1}{M_t} \cdot \alpha_{K,t} && (6.2)\\
\text{where } \alpha_{K,t} &= p(\mathbf{o}_1, \ldots, \mathbf{o}_t|\Omega) && (6.3)\\
&= \alpha_{K,t-1} \cdot b_K(\mathbf{o}_t) + \alpha_{K-1,t} && (6.4)\\
\text{and } M_t &= M_{t-1} \frac{K+t-1}{t} && (6.5)\\
\text{induced with } M_0 &= 1 && (6.6)\\
\text{and } \alpha_{k,0} &= 1, \alpha_{0,t} = 0, \forall k \leq K. && (6.7)
\end{aligned}
$$

Here, M_t denotes the number of paths through the OMM Ω if a time series of length t is observed, given a model with K model states, and $\alpha_{k,t}$ being the forward variable from the dynamic programming. Both M_t and $\alpha_{K,t}$ are recursively computed and only require the M_{t-1} and $\alpha_{K-1,t}$ values, respectively, as well as the newest observed sample $\mathbf{o}_t$.

Incremental classification can be realized analogously to non-incremental classification with a maximum likelihood decision: first, J class-specific models Ω are estimated from the data. An unknown time series then is assigned to the class associated with the model that yields the highest production likelihood $p(O_t|\Omega_j)$ of all models. In contrast to non-incremental classification, the production likelihoods are updated with each observed sample $\mathbf{o}_t$, and thus the classification decision.

6.1.2. Adaptive Learning

Our adaptive learning approach allows to train OMMs online, i.e., observation by observation while they are observed. The general idea of our online OMM training is similar to Liang and Klein [2009] [see also Neal and Hinton, 1998], who propose an incremental expectation maximization (iEM) procedure for probabilistic models. The training scheme of iEM subtracts the model adjustment from the previous example to add in the new one. Such a procedure can be straight-forwardly transfered to OMM training:

Let Ω be an OMM and Ω'_Δ the difference between this model and the model from the last iteration step. The algorithm to incorporate the time series O^i into the model online proceeds as follows:

1. Compute the responsibilities $r^i_{k,t}$ for Ω and O^i
2. Compute $\Omega_\Delta = \boldsymbol{\mu}_{\Delta,1}, \ldots, \boldsymbol{\mu}_{\Delta,K}$ according to

$$
\boldsymbol{\mu}_{\Delta,k} = \frac{\sum\limits_{t=1}^{T} r^i_{k,t} \cdot \mathbf{o}^i_t}{\sum\limits_{t=1}^{T} r^i_{k,t}}. \qquad (6.8)
$$

3. Update the model: $\Omega = \Omega + \Omega_\Delta - \Omega'_\Delta$
4. Replace old with new: $\Omega'_\Delta = \Omega_\Delta$
5. Repeat until convergence

	$NN_{\text{DTW-window}}$	SVM_{GDTW}		OMM	
data set	window size	C	γ	K	σ
Australian Sign Language Signs	0.5	5	0.002	34	1.5
Character Trajectories	1.0	5	0.002	171	0.47
Spoken Arabic Digits	0.4	1	0.008	39	1.125

Table 6.1.: Optimal hyperparamters found in cross validation for the benchmark data from the UCI repository.

6.2. Low Latency Recognition of Interaction Time Series

In order to determine the abilities of OMM-based classifiers for low latency recognition of time series data, we will, first, investigate how such classifiers perform as compared to standard approaches to time series classifications, both, in terms of generalization error, as well as running times.

In an experimental setup, we compare OMM classifiers to widely-used classification techniques on various time series data sets from the UCI machine learning repository [Frank and Asuncion, 2010], namely, the Australian Sign Language Signs [Kadous, 2002][1], the Character Trajectories Data Set[2], and the Spoken Arabic Digits Data Set[3]. Table 5.1 displays basic properties of these data sets. Besides OMM classifiers, we applied Nearest Neighbor Classifiers based on a Dynamic Time Warping distance function (NN_{DTW}) in two variants: (i) NN_{DTW}: here, the DTW function incorporates the complete dynamic programming scheme with all possible paths, (ii) $NN_{\text{DTW-window}}$: this classifier uses dynamic time warping with a reduced amount of warping paths, as well as (iii) SVM_{GDTW}: a SVM classifiers that uses a Gaussian Dynamic Time Warping kernels. See Section 5.1.1 for detailed descriptions of these classification approaches.

Results from these experiments are directly related to this thesis' major research question (RQ1).

6.2.1. Data Sets

The Australian Sign Language data set consists of 95 classes each containing 27 realizations of one particular sign. Each point in time is represented as 22 dimensional feature vectors that include the locations and orientations of both hands, as well as the bend measures of all fingers. The data was captured in nine sessions, of which we use the examples of the first seven sessions as training data, and data records from sessions eight and nine for testing.

The Character Trajectories data set consists of 2858 examples in 20 classes. Each class contains trajectories of one character that was written on a WACOM tablet by different human participants. A character is represented as a time series of three dimensional feature vectors (x/y coordinates and pen force). We randomly partitioned the data set in training and test sets of equal sizes.

[1] http://www.cse.unsw.edu.au/˜waleed/tml/data/

[2] http://archive.ics.uci.edu/ml/datasets/Character+Trajectories

[3] http://archive.ics.uci.edu/ml/datasets/Spoken+Arabic+Digit

data set	$\mathrm{NN}_{\mathrm{DTW}}$	$\mathrm{NN}_{\mathrm{DTW\text{-}window}}$	$\mathrm{SVM}_{\mathrm{GDTW}}$	OMM
Australian Sign Language Signs	≈ 0.039	≈ 0.039	≈ 0.007	≈ 0.019
Character Trajectories	≈ 0.064	≈ 0.051	≈ 0.041	≈ 0.109
Spoken Arabic Digits	≈ 0.055	≈ 0.027	≈ 0.008	≈ 0.028

Table 6.2.: Test set error rates for time series data sets from the UCI machine learning repository.

The Spoken Arabic Digits data set provides 8800 examples in 10 classes that are represented by time series of 13 dimensional mel frequency cepstral coefficients. This data set provides dedicated training and test data sets.

6.2.2. Experiments

We identified optimal hyperparameter values as described in Section 5.2. For OMMs, we chose 10 equidistant values for the number of states K depending on the (average) time series length T with rounded values from $\{\frac{T}{10}, \frac{2 \cdot T}{10}, \ldots, T\}$. The set of values for the standard deviation parameter σ was equal in all cases: $\sigma \in \{2 \cdot 0.75^y | y = 1, \ldots, 10\}$.

Similarly, for SVM classifiers, we chose the penalty constant $C \in \{10^y, 5 \cdot 10^y | y = 0, \ldots, 4\}$ and the GDTW kernel parameter $\gamma \in \{0.5^y | y = 0, \ldots, 9\}$.

For $\mathrm{NN}_{\mathrm{DTW\text{-}window}}$ classifiers, we evaluated a window length factor from $\{0.1, 0.2, \ldots, 1.0\}$. Please note that a window length factor of 1.0 is equal to a $\mathrm{NN}_{\mathrm{DTW}}$ classifier that incorporates the complete DTW path search space. The $\mathrm{NN}_{\mathrm{DTW}}$ classifiers do not incorporate any hyperparameters.

6.2.3. Results and Discussion

Table 6.2 shows the performance results of our evaluation in terms of error rates on the test sets. The optimal hyperparameters found in cross validation are denoted in Table 6.1. In general, all methods show sufficient accuracy, whereby the discriminative kernel-based $\mathrm{SVMs}_{\mathrm{GDTW}}$ systematically outperform OMMs and the Nearest Neighbor approach. On two out of three data sets OMMs outperform $\mathrm{NN}_{\mathrm{DTW}}$ classifiers while a contrary result can be observed on the third data set.

The superior performance of the kernel-based $\mathrm{SVM}_{\mathrm{GDTW}}$ comes at the cost of substantially increased computational demands. For classification of an unseen example, kernel-based approaches often require the calculation of a large number of kernel functions. Table 6.3 shows the average running times for classification of a single example for the three methods. In comparison to $\mathrm{SVM}_{\mathrm{GDTW}}$, OMMs demonstrate a speed-up factor between ≈ 25 (Character Trajectories) and ≈ 339 (Spoken Arabic Digits). In contrast to $\mathrm{SVM}_{\mathrm{GDTW}}$ and $\mathrm{NN}_{\mathrm{DTW}}$, the short running times of OMMs underline their suitability for real-time applications as required in many interaction scenarios. Note that the running times of $\mathrm{SVM}_{\mathrm{GDTW}}$ depend on the sparseness of the classifier, i.e., how many support vectors are actually used.

In terms of this thesis' **major research questions** (Section 2.1) the results of this

data set	NN_{DTW}	$\text{NN}_{\text{DTW-window}}$	SVM_{GDTW}	OMM
Australian Sign Language Signs	$\approx 5.75s$	$\approx 3.06s$	$\approx 5.71s$	$\approx 0.13s$
Character Trajectories	$\approx 10.10s$	$\approx 10.10s$	$\approx 5.86s$	$\approx 0.23s$
Spoken Arabic Digits	$\approx 6.80s$	$\approx 3.14s$	$\approx 3.39s$	$\approx 0.01s$

Table 6.3.: Average running times in seconds for classification of one test example.

experiment show that OMM-based classifiers are able to recognize interaction data with high accuracy **(Q1)**. In contrast to highly accurate kernel machines, the OMM algorithm is substantially more computational efficient, which allows low latency response times in interaction scenarios **(Q2)**.

An additional shortcoming of kernel machines is that in general the model is not easily interpretable in terms of characteristic features in data space. Such features may provide additional insights into the underlying data distribution including regularities and variations of relevant patterns. In Chapter 7, we will investigate if and how all estimated OMM parameters can directly be inspected in terms of a prototype representation.

6.3. Robust Recognition: Incomplete Data

In the previous section, we found that OMM-based classifiers are able to classify and recognize interaction data rapidly and with high accuracy. However, in interaction scenarios, robustness towards unpredictable errors is a crucial prerequisite.

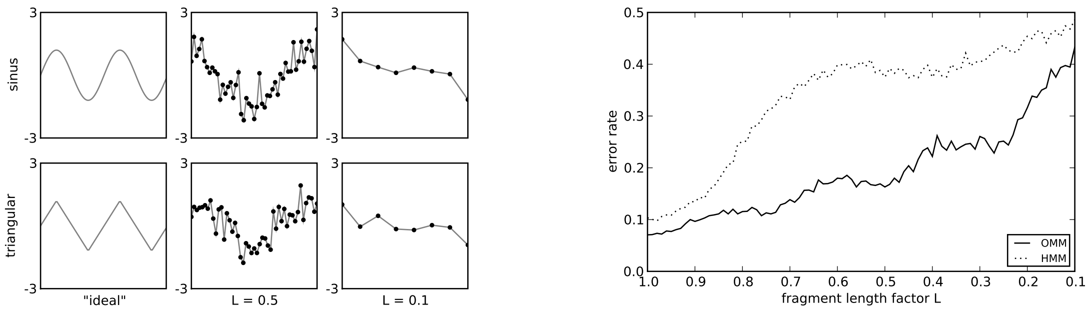
To test if OMM classifiers are robust towards incomplete observations, we will evaluate OMM-based classifiers, first, with an artificial simulation data where we control all aspects of the processed data set. Second, we assess the robustness properties of OMM classifiers in comparison to standard classification methods in 20 benchmark data sets for time series classification.

6.3.1. Data Set

We designed a simulation experiment in which we evaluated the classification accuracy with fragmented test examples of decreasing length.

For this purpose, we created a binary classification task containing a class of sine waves and a class of triangular waves, each covering two complete periods. The time series lengths follow a truncated Gaussian distribution with $\mu = 60$, $\sigma = 20$ and lower and upper bounds of 20 and 100, respectively. Additionally, we apply Gaussian noise to the signals so that the resulting time series have a signal-to-noise ratio of $SNR = 5$. The simulation data set consists of 2200 time series and is partitioned into a training set of 200 time series (100 for each class) and a test set of the remaining 2000 time series (1000 for each class).

Furthermore, we created additional test sets of decreasingly shorter fragments out of the original test set time series. The length of each fragment is relative to the original length of the time series according to a *fragment length factor* L between $L = 1.0$ and $L = 0.1$. For example, if the length of a test time series is $T = 100$, then its fragment length for $L = 0.5$



(a) Examples from the simulation data set. Each row shows one class while the columns show time series fragments for different values of the fragment length factor L. The first column shows ideal noise-free waves as used for example generation.

(b) Error rates for fragmented test time series from the simulation data set. The y-axes denotes the error rate while the x-axes corresponds to time series fragments of decreasing length according to the fragment length factor L.

Figure 6.1.: These plots show example data from the simulation data sets (a) and related classification results (b).

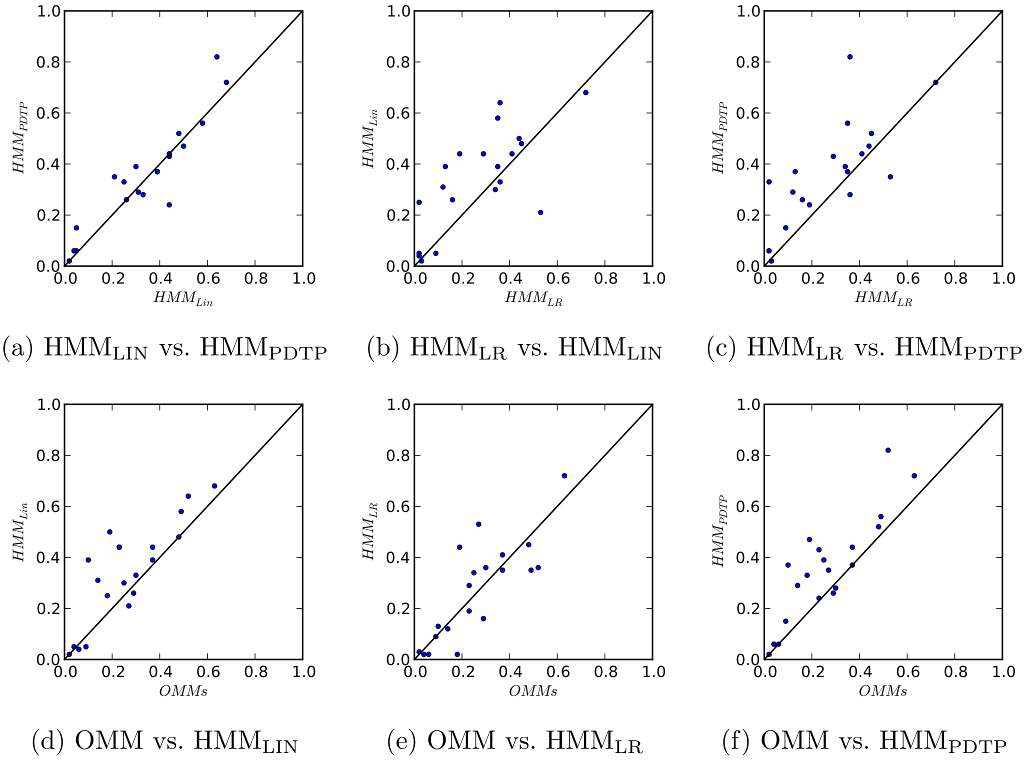
is $T_{L=0.5} = 50$. In this study, L decreases in steps of 0.01 to provide 91 different test data sets with correspondingly shorter fragments. The location of the fragment is randomly chosen for each time series O and fragment length factor L. Figure 6.1a shows examples from the simulation data set for different values of L together with ideal noise-free waves used as the basis for data set generation.

6.3.2. Experiments

We compared OMM-based classifiers with an HMM-based approach (cf. the HMM$_{\mathrm{LR}}$ approach as described in Section 5.1.1), with similar model properties as the used OMM classifier: (i) a flexible left-to-right topology, (ii) emissions that are modeled as Gaussian's with a global standard deviation parameter σ that is used as a hyperparameter. However, in contrast to OMMs, HMMs$_{\mathrm{LR}}$ require the estimation of transition probabilities from the data. We selected optimal classifiers by means of 5-fold cross validation on the training data (cf. Section 5.2). Subsequently, we tested the performance of the arising OMM- and HMM$_{\mathrm{LR}}$-classifiers with all 91 test data sets corresponding to the decreasing L factor values.

6.3.3. Results and Discussion

The results of the simulation experiments are summarized in Figure 6.1b. The plot reflects the dependency of the error rate on decreasing values of the fragment length factor L. Here, OMMs clearly outperform HMMs$_{\mathrm{LR}}$ in terms of correctly classified fragments. Although OMMs and HMMs$_{\mathrm{LR}}$ perform almost equally on full-length test time series, the error rate of HMMs$_{\mathrm{LR}}$ increases rapidly on the fragmented data. While both models provide an error rate of less than 10% for full test time series, the error rates differ by ≈ 0.2 at $L = 0.53$. In particular, OMMs provide an error rate below 0.2 for fragments of less than half the



(a) HMM_{LIN} vs. HMM_{PDTP} (b) HMM_{LR} vs. HMM_{LIN} (c) HMM_{LR} vs. HMM_{PDTP}

(d) OMM vs. HMM_{LIN} (e) OMM vs. HMM_{LR} (f) OMM vs. HMM_{PDTP}

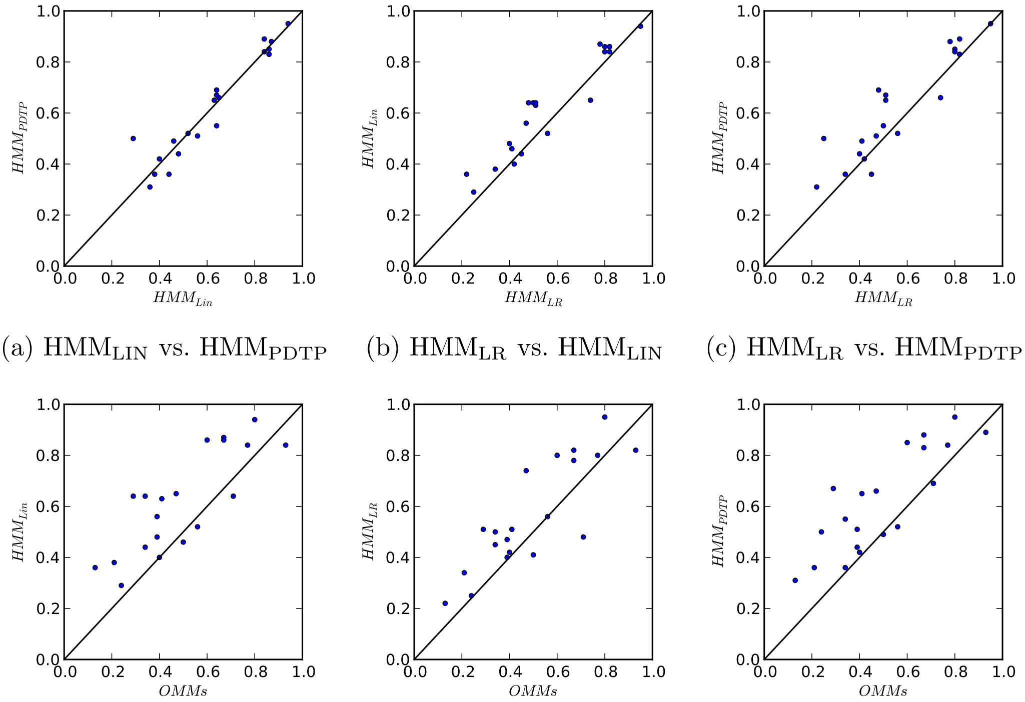
Figure 6.2.: Comparative scatter plots of the error rates associated with the experimental results for Ordered Means Models (OMM), Hidden Markov Models with left-to-right topology (HMM_{LR}), with linear model topology (HMM_{LIN}), and with pre-defined transition probabilities (HMM_{PDTP}). These plots illustrate the results for the original (complete) test data. Each marker corresponds to one data set; the x- and y-axes denote the error rates for the specified models.

length of the original data. In contrast, HMMs_{LR} do not reach this accuracy level even for fragments with a length above 80% of the full time series length. While OMMs do not exceed the 0.3 error rate before $L = 0.2$, HMMs_{LR} already reach that error level for $L = 0.75$. These results indicate that HMMs_{LR} are more susceptible to fragmented data than OMMs.

According to the major research questions of this thesis, the results of this experiment are a first indicator that OMMs are able to recognize incomplete data with higher accuracy as compared to HMMs **(Q3)**.

6.4. Robust Recognition: Influence of Transition Probabilities

The results from the simulation data experiment suggest that OMM-based classifiers are well suited for classification of incomplete data. To further support these findings, we evaluated OMM-based classifiers with real-world time series data sets from various data domains (cf. Section 5.6).



(a) $\mathrm{HMM_{LIN}}$ vs. $\mathrm{HMM_{PDTP}}$ (b) $\mathrm{HMM_{LR}}$ vs. $\mathrm{HMM_{LIN}}$ (c) $\mathrm{HMM_{LR}}$ vs. $\mathrm{HMM_{PDTP}}$

(d) OMM vs. $\mathrm{HMM_{LIN}}$ (e) OMM vs. $\mathrm{HMM_{LR}}$ (f) OMM vs. $\mathrm{HMM_{PDTP}}$

Figure 6.3.: Comparative scatter plots of the error rates associated with the experimental results for Ordered Means Models (OMM), Hidden Markov Models with left-to-right topology ($\mathrm{HMM_{LR}}$), with linear model topology ($\mathrm{HMM_{LIN}}$), and with pre-defined transition probabilities ($\mathrm{HMM_{PDTP}}$). These plots illustrate the results for he comparisons for the half-length fragments generated from the testing data. Again, each marker corresponds to one data set; the x- and y-axes denote the error rates for the specified models.

6.4.1. Data Sets

We tested the proposed models with common benchmark data sets for time series classification from the UCR time series repository [Keogh et al., 2006]. The repository contains 20 data sets from various research fields, each providing dedicated training and test sets with time series of equal length. The data dimensionality of all data sets is $d = 1$. Table 5.1 in Section 5.6 displays basic properties of the UCR benchmark data sets.

6.4.2. Experiments

To test the influence of the transition probabilities on the robustness of the proposed models, we designed an experiment, which corresponds to the observation of incomplete data — similar to the one described in Section 6.3. Here, fragments of half the length (50%) of the original time series are used. The beginning of the fragment was chosen randomly for each time series. For evaluation, we tested the resulting fragments with the classifiers obtained from the original training data.

model	fragment length factor							
	$L = 1.0$				$L = 0.5$			
	rank	wins−losses	wins	losses	rank	wins−losses	wins	losses
OMMs	1	22	38	16	1	36	47	11
HMMs$_{\text{LR}}$	1	22	40	18	2	9	33	24
HMMs$_{\text{LIN}}$	3	-17	19	36	3	-19	19	38
HMMs$_{\text{PDTP}}$	4	-27	13	40	4	-26	15	41

Table 6.4.: Results from a ranking test for Ordered Means Models (OMMs), Hidden Markov Models with left-to-right topology (HMMs$_{\text{LR}}$), Hidden Markov Models with linear model topology (HMMs$_{\text{LIN}}$), and Hidden Markov Models with pre-defined transition probabilities (HMMs$_{\text{PDTP}}$) on the benchmark data sets. The second column (”$L = 1.0$”) shows the ranking for complete time series, the third column (”$L = 0.5$”) the ranking for fragments of 50% length of the original data.

Since OMMs are defined without any transition probabilities, we evaluated the generalization performance of four model types to investigate the influence of transition probabilities:

- How does the absence of transition probabilities affect the generalization performance on benchmark data?
- How does the model robustness depend on transition probabilities when incomplete (fragmented) observations occur?

We compared OMM-based classifiers with three HMM-based approaches, each of which follow a different modeling. For comparison, we chose (i) HMM$_{\text{LR}}$, (ii) HMM$_{\text{LIN}}$, and (iii) HMM$_{\text{PDTP}}$. Please see Section 5.1.1 for a detailed description of these methods.

In the process of hyperparameter selection by means of 5-fold cross validation (cf. Section 5.2), we chose ten equidistant values for the number of states K depending on the time series length T with rounded values from $\{\frac{T}{10}, \frac{2 \cdot T}{10}, \ldots, T\}$. The set of values for the standard deviation parameter σ is equal in all cases: $\sigma \in \{1.5 \cdot 0.75^y | y = 0, \ldots, 9\}$. These values are the same for all evaluated models.

As a measure for comparison of the generalization performance, we used the empirical classification error on test sets. We summarized the results of both complete and fragmented data in terms of a ranking test as used in the WEKA toolbox [Hall et al., 2009]. In a ranking test, the error rates across all data sets are compared for each pair of methods. In case one particular method outperforms another one, this is counted as a ’win’, the opposite case is counted as a ’loss’. All methods are then ranked according to the difference between the number of wins and losses.

data set	OMMs		HMMs_{LR}		HMMs_{LIN}		$\text{HMMs}_{\text{PDTP}}$	
	K	σ	K	σ	K	σ	K	σ
50words	162	0.84	135	0.47	108	0.84	135	0.84
Gun_Point	135	0.15	120	0.15	45	0.36	150	0.2
OSULeaf	427	0.63	214	0.47	342	0.47	214	1.13
Two_Patterns	51	0.63	26	0.27	26	1.13	26	0.27
Adiac	88	0.27	106	0.27	106	0.2	123	0.36
ECG200	58	0.2	96	0.27	77	0.84	77	1.5
wafer	30	0.36	122	0.47	106	1.5	137	1.13
Beef	47	0.63	94	0.27	94	0.47	141	1.13
FaceAll	79	0.63	131	0.36	105	0.63	79	0.84
Lighting2	573	0.84	573	0.84	255	1.13	510	1.5
SwedishLeaf	90	0.36	115	0.2	102	0.36	90	0.27
yoga	298	0.36	85	0.47	128	0.84	85	0.84
CBF	13	1.5	115	0.2	38	0.15	13	1.5
FaceFour	175	0.84	315	0.36	280	0.27	280	0.47
Lighting7	191	0.47	287	1.5	96	1.13	96	0.63
synthetic_control	24	0.84	36	0.47	42	0.84	30	0.27
Coffee	200	1.5	86	0.63	86	0.27	200	0.27
fish	463	0.47	232	0.36	370	0.2	370	0.36
OliveOil	228	1.5	513	0.11	456	0.11	570	0.47
Trace	28	0.15	275	0.11	138	0.2	138	0.2

Table 6.5.: The best hyperparameters K (number of states) and σ (global standard deviation) from cross validation for Ordered Means Models (OMMs), Hidden Markov Models with left-to-right topology (HMMs_{LR}), with linear model topology (HMMs_{LIN}), and with pre-defined transition probabilities ($\text{HMMs}_{\text{PDTP}}$) on the benchmark data sets from Keogh et al. [2006].

6.4.3. Results and Discussion

Table 6.6 shows the error rates for all data sets and evaluated models. Figures 6.2 and 6.3 illustrate these results with comparative scatter plots of the error rates for complete and fragmented time series, respectively. Each marker corresponds to one data set; the x- and y-axes denote the error rates for the specified models.

The corresponding Table 6.4 summarizes the performance results from our experiments in terms of a ranking test as used in the WEKA toolbox [Hall et al., 2009]. The second column of Table 6.4 shows the results for complete time series ($L = 1.0$), while the third column gives an overview of the performance values on fragments of 50% length of the original data ($L = 0.5$).

For complete time series, the models with left-to-right topology (OMM and HMM_{LR}) both rank at first position, while the models with linear model topology (HMM_{LIN} and HMM_{PDTP}) rank at position three and four. In the case of simulated missing data, OMM exclusively rank at first position, while the HMM_{LR} rank as second, and again followed by HMM_{LIN} and HMM_{PDTP}.

	complete				fragments			
data set	OMMs	HMMs$_{\text{LR}}$	HMMs$_{\text{LIN}}$	HMMs$_{\text{PDTP}}$	OMMs	HMMs$_{\text{LR}}$	HMMs$_{\text{LIN}}$	HMMs$_{\text{PDTP}}$
50words	**0.25**	0.34	0.3	0.39	0.93	**0.82**	0.84	0.89
Gun_Point	0.18	**0.02**	0.25	0.33	**0.24**	0.25	0.29	0.5
OSULeaf	0.49	**0.35**	0.58	0.56	0.71	**0.48**	0.64	0.69
Two_Patterns	0.04	**0.02**	0.05	0.06	0.5	**0.41**	0.46	0.49
Adiac	**0.63**	0.72	0.68	0.72	**0.8**	0.95	0.94	0.95
ECG200	0.23	**0.19**	0.44	0.24	**0.34**	0.5	0.64	0.55
wafer	**0.1**	0.13	0.39	0.37	**0.39**	0.47	0.56	0.51
Beef	0.48	**0.45**	0.48	0.52	0.56	0.56	**0.52**	**0.52**
FaceAll	**0.23**	0.29	0.44	0.43	**0.67**	0.78	0.87	0.88
Lighting2	0.3	0.36	0.33	**0.28**	**0.34**	0.45	0.44	0.36
SwedishLeaf	**0.37**	0.41	0.44	0.44	**0.67**	0.82	0.86	0.83
yoga	0.37	**0.35**	0.39	0.37	**0.39**	0.4	0.48	0.44
CBF	0.06	**0.02**	0.04	0.06	**0.13**	0.22	0.36	0.31
FaceFour	0.14	**0.12**	0.31	0.29	**0.41**	0.51	0.63	0.65
Lighting7	0.27	0.53	**0.21**	0.35	**0.47**	0.74	0.65	0.66
synth_control	**0.02**	0.03	**0.02**	**0.02**	**0.21**	0.34	0.38	0.36
Coffee	0.29	**0.16**	0.26	0.26	**0.4**	0.42	**0.4**	0.42
fish	**0.19**	0.44	0.5	0.47	**0.6**	0.8	0.86	0.85
OliveOil	0.52	**0.36**	0.64	0.82	**0.77**	0.8	0.84	0.84
Trace	0.09	0.09	**0.05**	0.15	**0.29**	0.51	0.64	0.67

Table 6.6.: Test set error rates for Ordered Means Models (OMMs), Hidden Markov Models with left-to-right topology (HMMs$_{\text{LR}}$), with linear model topology (HMMs$_{\text{LIN}}$), and with pre-defined transition probabilities (HMMs$_{\text{PDTP}}$) on the benchmark data sets from Keogh et al. [2006]. The best error rates obtained for each data set are printed in bold.

In addition to the ranking test, we also performed a Wilcoxon signed rank test to measure the significance of our findings. Table 6.7 shows the p-values from paired, two-sided tests applied to the error rates of complete time series (Table 6.7a) and fragmented data (Table 6.7b).

Corresponding to the results from the ranking test, the signed rank test does not indicate a performance difference between OMMs and HMMs$_{\text{LR}}$ for complete time series ($p = 0.936$). However, the performance of OMMs is superior on fragmented time series as indicated by the ranking test is significant ($p = 0.018$) given a widely used confidence interval of 5%. Combining the results from both ranking and signed rank test, models with a left-to-right topology consistently outperform models with a linear topology, while no evidence for a performance difference between models with a linear topology is given.

The comparison of the models with linear topology gives no evidence for different performances in both scenarios. The signed rank test on the error rates of HMMs$_{\text{LIN}}$ and

	HMM_{LR}	HMM_{LIN}	HMM_{PDTP}
OMM	0.936	0.005	0.001
HMM_{LR}		0.037	0.006
HMM_{LIN}			0.407

(a) P-values from signed rank tests for the error rates from the models on complete time series.

	HMM_{LR}	HMM_{LIN}	HMM_{PDTP}
OMM	0.018	0.001	0.001
HMM_{LR}		0.004	0.013
HMM_{LIN}			0.844

(b) P-values from signed rank tests for the error rates from the models on fragmented time series.

Table 6.7.: P-values from a signed rank test for error rates in complete time series observations (a) and fragments of half length (b) from the UCR time series classification/clustering benchmark repository [Keogh et al., 2006]

$\text{HMMs}_{\text{PDTP}}$ results in a p-value of 0.4072 for complete time series and in a p-value of 0.8444 for the fragments. The models with left-to-right topology consistently outperform the models with linear topology as indicated by the results of the signed rank test and the ranking test.

To analyze the practical relevance of the algorithm complexity of the proposed models, we also measured the running times for the whole evaluation procedure in terms of single core CPU time on an Intel Xeon CPU with 2.5GHz. The total running times for OMMs, HMMs_{LR}, HMMs_{LIN}, and $\text{HMMs}_{\text{PDTP}}$ accumulate to ≈ 82, ≈ 452, ≈ 172, and ≈ 101 days, respectively.

Last, Table 6.5 on page 69 shows the optimal hyperparameter values for all benchmark data sets and models. Here, K denotes the number of states and σ the standard deviation associated with the lowest cross validation error rate. OMMs yield the smallest models in terms of the number of states in 11 cases, while HMMs_{LR}, HMMs_{LIN}, and $\text{HMMs}_{\text{PDTP}}$ provide the smallest models in four, five, and six cases, respectively.

With regard to this thesis' major research questions (Section 2.1), these results indicate that: First, the explicit modeling of transition probabilities does not seem to provide a general advantage, although, in particular cases the implied structure may result in a better generalization performance. In general, HMMs as well as OMMs are well suited to represent time-series and both methods could reach very low error rates on benchmark classification problems **(Q1)**. Second, exclusion of transition probabilities can enormously reduce the computational needs. This enables the application of OMMs in situations with limited computational resources. On the other hand, for a given limit on runtime, OMMs allow to represent a higher resolution of temporal dynamics using an increased number of states, i.e., a higher density of reference vectors **(Q2)**. Third, the implicit modeling of the time series length distribution by transition probabilities has an impact on the robustness of the learned model with respect to fragmented data **(Q3)**.

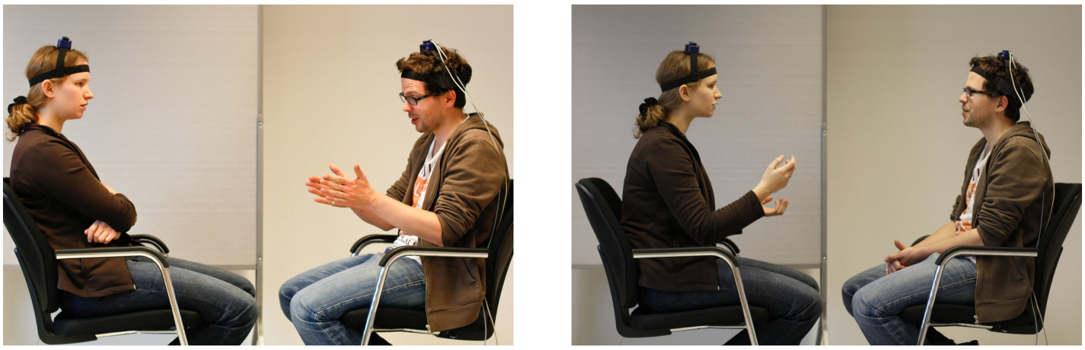


Figure 6.4.: Scenario for data acquisition. Both participants are wearing a MT9 inertial sensor to measure head motions.

6.5. Scenario: Conversational Head Gestures

Head gestures are important for human communication, e.g., Munhall et al. [2004] showed that head movements improve the perception of syllables in Japanese. In order to make conversation with virtual agents and social robots more lifelike, it might be useful to allow recognition and reproduction of head gestures [Cassell et al., 1994]. To allow agents to perform head gestures (i) at appropriate timing, and (ii) in a suitable pace it is initially necessary to investigate the appearance of head gestures in human-human interaction. For such research questions, typically, video data are manually annotated. This costly and time-consuming process can be dramatically simplified by a combination of machine-learning methods and head tracking techniques. Possible approaches are described in detail by Vatavu et al. [2005] who states that vision-based approaches [e.g. Morency et al., 2002] benefit from an unobtrusive measurement but are dependent on constant lighting conditions and on a full view of the interactants' faces. In addition, high computing power is necessary to analyze the recorded video data streams.

In this experiment, we chose a sensor-based approach for recognition of head gesture data. Therefore, we mounted motion sensors on the participant's head, which allows independence of lighting conditions and almost unrestricted mobility. In addition, a motion sensor is easy-to-apply and usable without any calibration.

To examine such a sensor-based head gesture recognition system, we accomplished a study with 10 subjects in comparatively natural setups. We abdicated any sensor preparations and adjustments, and analyzed the recorded data with regards to separability and robustness towards incomplete observations.

6.5.1. Data Set

We recorded the experimental data by means of a Xsens MT9 inertial sensor[4] (we only used the gyroscopes with 3 DoF rate-of-turn). With this sensor attached to the top of the participants' heads, we recorded the head movements of 10 subjects during dyadic conversation (see Figure 6.4). All subjects were German native speakers and briefly informed about the purpose of the experiment. The conversation was videotaped by a scene camera

[4] www.xsens.com

	Number of Events for					
Participant	Nod	Shake	Tilt	Look	Length	Duration
1	27	17	0	2	0.98s	11 m
2	22	14	0	2	1.1s	11 m
3	33	1	0	0	1.39s	15 m
4	22	1	4	3	1.09s	15 m
5	35	67	2	37	1.4s	33 m
6	27	38	6	26	1.2s	33 m
7	77	81	15	47	1.46s	27 m
8	15	16	1	49	1.04s	27 m
9	122	37	13	65	1.42s	43 m
10	67	33	3	79	1.17s	43 m
$\sum$	447	305	44	310		

Table 6.8.: This table denotes the number of head movement samples per participant: the participant's number, the medium length of event (in seconds), overall duration of the measurement (in minutes).

and we disrupted the experiment when the conversation became stagnant. We synchronized the sensor data with the scene camera video and annotated the head movements in ELAN [Hellwig and Uytvanck, 2004]. We then left out all weak annotations and sliced the head motion data into segments that are related to the four most frequent occurring head movement annotations: *nod*, *shake*, *tilt* (occurred as a gesture of uncertainty), and *look* (sideways). Table 6.8 illustrates the resulting samples.

We randomly partitioned the complete data set into a train and a testing data subset. All data were recorded at 33Hz and normalized to zero-mean and unit variance.

6.5.2. Experiments

To estimate the accuracy of OMMs for head gesture recognition, we tested these four head movement classes with three experiments based on the obtained dataset. We examined:

1. Are OMM classifiers suitable for recognition of head gestures that are observed by means of goniometers?
2. Which combinations of head gesture classes are easily separable by means of an OMM-based classifier, and which combinations are not?
3. What influence do incomplete observations have on the recognition capabilities of OMM-based classifiers in this context?

In order to answer these research questions, we conducted the following experiments. First, we applied a standard 5-fold cross validation procedure on the training data to obtain optimal OMM hyperparameters (K, σ) for the given recognition task. We use the hyperparameters found in this process in all following experiments. Afterward, we tested the generalization properties of the classifier by means of the remaining testing data.

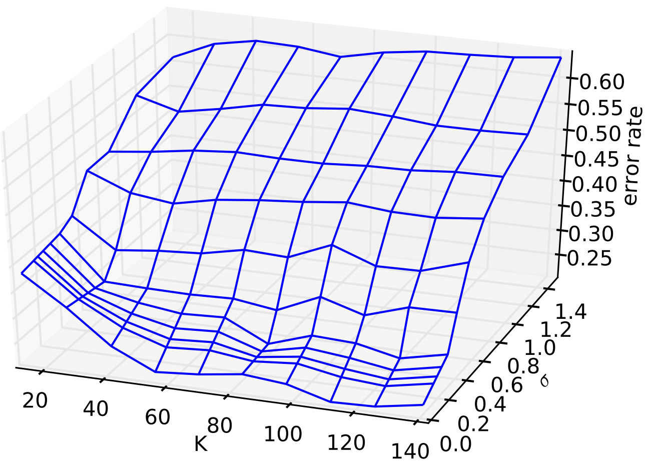


Figure 6.5.: This figure illustrates the results from the cross validation procedure for all hyperparameter pairs (K, σ) for the head gesture data set.

To answer the second research question, we evaluated the obtained classifier with all elements in the power set of the testing data's classes with cardinality greater than 1, i.e., we used all feasible class combinations for testing:

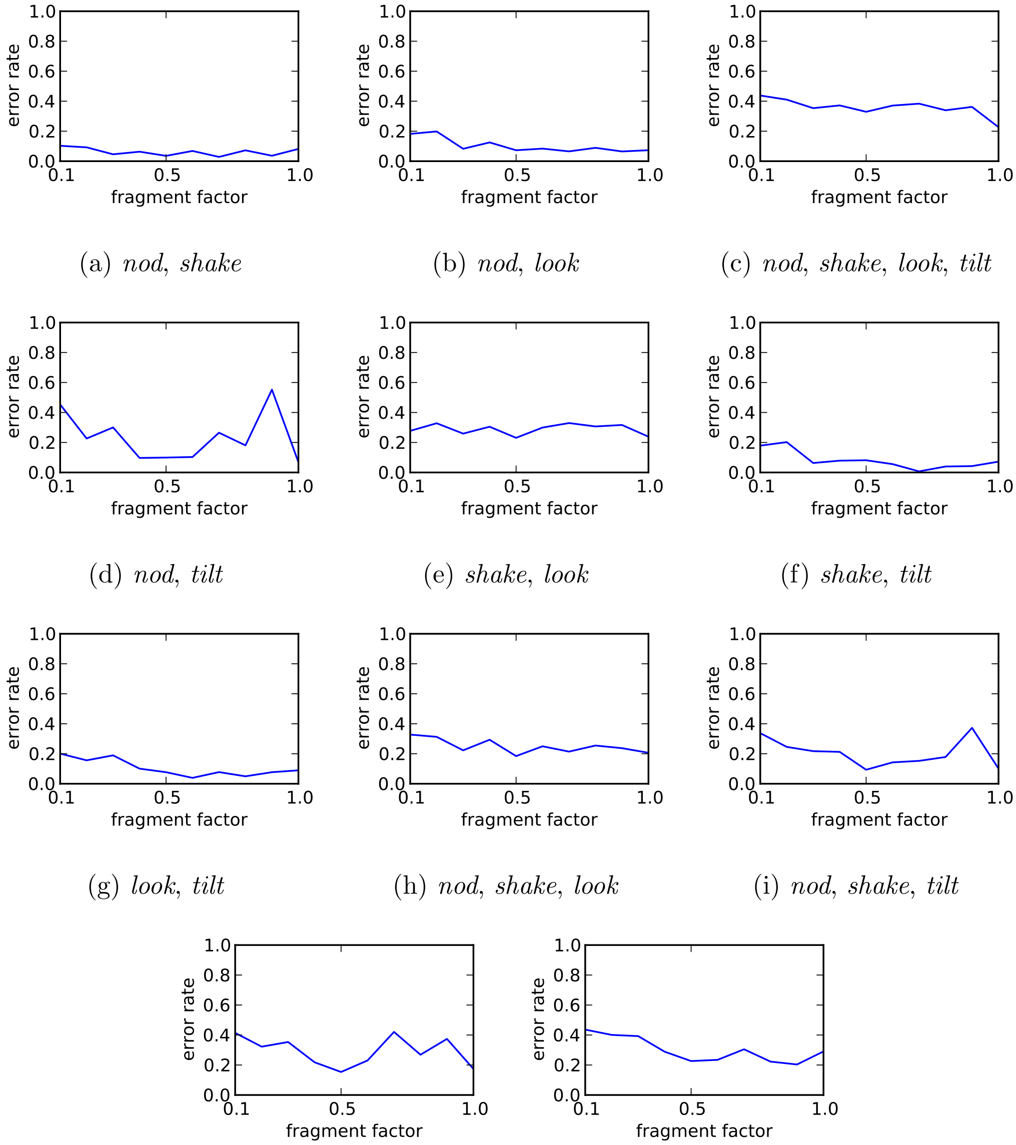
$$\begin{aligned} Y = &\ \{\textit{nod}, \textit{shake}, \textit{look}, \textit{tilt}\} \\ \mathcal{P}_{>1}(Y) = &\ \{\{\textit{nod}, \textit{shake}\}, \{\textit{nod}, \textit{look}\}, \{\textit{nod}, \textit{tilt}\}, \{\textit{shake}, \textit{look}\}, \{\textit{shake}, \textit{tilt}\}, \\ &\ \{\textit{look}, \textit{tilt}\}, \{\textit{nod}, \textit{shake}, \textit{look}\}, \{\textit{nod}, \textit{shake}, \textit{tilt}\}, \{\textit{nod}, \textit{look}, \textit{tilt}\}, \\ &\ \{\textit{shake}, \textit{look}, \textit{tilt}\}, \{\textit{nod}, \textit{shake}, \textit{look}, \textit{tilt}\}\} \end{aligned}$$

For investigating the influence of incomplete head gesture observations, we tested the classifier with fragmented test examples of decreasing length. The length of each fragment is relative to the original length of the head gesture according to a *fragment length factor* L between $L = 1.0$ and $L = 0.1$, similar to the procedure described in Section 6.3. In this experiment, we used 10 values for the fragment length factor $L \in \{0.1, 0.2, \ldots, 1.0\}$.

During cross validation (cf. Section 5.2), we chose 10 different values for the number of OMM states K with $K \in \{14, 28, 42, 56, 70, 84, 98, 112, 126, 141\}$. The set of values for the global standard deviation σ was $\sigma \in \{1.5 \cdot 0.75^x | x = 0, \ldots, 9)\} \cup \{0\}$.

6.5.3. Results and Discussion

The results of the cross validation experiment reveals error rates between ≈ 0.21 and ≈ 0.64 with an average of ≈ 0.38. Figure 6.5 illustrates the cross validation landscape. The plot indicates that the recognition rate of the given head gestures increases with lower σ values. And indeed, the optimal hyperparameter pair found in this experiment is $(K, \sigma) = (112, 0)$, i.e., the OMMs comprises 112 model states combined with a Viterby training. While evaluating the testing data, these hyperparameters yield an error rate of ≈ 0.23.



(a) *nod*, *shake*

(b) *nod*, *look*

(c) *nod*, *shake*, *look*, *tilt*

(d) *nod*, *tilt*

(e) *shake*, *look*

(f) *shake*, *tilt*

(g) *look*, *tilt*

(h) *nod*, *shake*, *look*

(i) *nod*, *shake*, *tilt*

(j) *nod*, *look*, *tilt*

(k) *shake*, *look*, *tilt*

Figure 6.6.: This figure illustrates the influence of incomplete head gesture observations on the recognition accuracy of the evaluated head gesture class sets. The x-axes of the plots denote the fragment length factor, i.e., the proportional fragment length as compared to the full head gesture length. The y-axes denote the error rate.

head gestures	error rate	running time
nod, *shake*	0.06	0.02 s
nod, *look*	0.09	0.01 s
nod, *tilt*	0.06	0.02 s
shake, *look*	0.28	0.01 s
shake, *tilt*	0.07	0.02 s
look, *tilt*	0.07	0.01 s
nod, *shake*, *look*	0.21	0.02 s
nod, *shake*, *tilt*	0.1	0.02 s
nod, *look*, *tilt*	0.12	0.02 s
shake, *look*, *tilt*	0.3	0.02 s
nod, *shake*, *look*, *tilt*	0.23	0.03 s

Table 6.9.: This table denotes the error rates and average running times for classification of one example for the classification experiment of head gestures.

real gesture	classified gesture			
	nod	*shake*	*tilt*	*look*
nod	**0.87**	0.05	0.04	0.04
shake	0.04	**0.86**	0.03	0.07
tilt	0.02	0.23	**0.62**	0.08
look	0.1	0.41	0	**0.48**

Table 6.10.: Here, the confusion matrix for the classification of the head gesture data set is denoted. Please note, that we report the accuracy, i.e., the rate of mis-classified examples.

If we analyze the influence of the selected head gestures classes (see Table 6.9), the lowest error rate of ≈ 0.06 is achieved when *nod* vs. *shake*, and *nod* vs. *tilt*, respectively, are used as class sets, closely followed by *shake* vs. *tilt*, and *look* vs. *tilt* with an error rate of ≈ 0.07. In case of three classes, the lowest error rate of ≈ 0.1 yields a classifier trained for recognition of *shake*, *nod*, *tilt*. The highest error rate of ≈ 0.28 occurred in case of *shake* vs. *look*. If all classes are used, the classification error rate reaches ≈ 0.23. The results indicate that OMM-based classifiers heavily depend on the chosen class set. In particular, class sets that include both, *shake* and *look*, classes suffer from mis-classifications, probably caused by confusing the head gestures from these two classes. This assumption is further underlined by the confusion matrix displayed in Table 6.10. Based on this matrix, we can see that $\approx 41\%$ of all *look* gestures are inadequately recognized as *shake* movements. However, the recognition performance in the opposite direction is substantially better: only 7% of all *shake* head gestures are classified as *look*. This might be due to the fact that *look* gestures can be described as fragments of *shake* movements: while the head moves several times from left to right and back when a *shake* gesture is performed, a typical *look* gesture stops after only one "look" to the left or to the right.

Figure 6.6 illustrates the influence of incomplete head gesture observations on the recognition accuracy. The x-axes of the plots denote the fragment length factor, i.e., the proportional fragment length as compared to the full head gesture length. The y-axes denote the error rate. In contrast to the findings from the results from the experiments with fragmented data as presented in Section 6.3, here, OMM-based classifiers are able to recognize even short fragments of head gestures with low error rate. In case that all four head gesture classes are used, an average fragmented head gesture only reaches an error rate that is ≈ 0.06 points higher than the error rate yield by complete head gestures. And fragments with a fragment length factor $L = 0.1$, i.e., fragments that cover only 10% of the complete observations length' reach an average recognition error rate that is ≈ 0.16 higher as compared to complete gestures. Even though the classification accuracy of complete head gesture observations is substantially higher as compared to the much shorter fragments, the good performance of fragmented head gestures indicates that it might be useful to only apply the beginning of an observation for recognition. E.g., when an interactive system demands very short response times, the required running time can be dramatically reduced by using only the first 10% or 20% of an observation for classification.

According to the major research questions of this thesis, these results indicate that OMM-based classifiers are able to recognize head gesture with high accuracy **(Q1)** and low response times **(Q2)**. In particular in combination with robust recognition of fragmented head gestures **(Q3)**, the already low response times are once more dramatically reduced — with only little loss of recognition performance.

6.6. Scenario: Finger Pressure Patterns in Playing Musical Instruments

Teaching, performing, and exercising instruments are everyday tasks for musicians. Such activities can greatly benefit from sensing technology. Sensors measurements can support instrument teachers by exposing otherwise hidden mistakes in the student's handling of an instrument. And students can benefit from such technology in their exercises, e.g., by preventing fatigue or malpositions — factors that often lead to ailments and pain.

In this study, we used a sensor-based approach to monitor bowing gestures of violin players. Thereby, our sensing system is a (i) small, lightweight, and wireless, (ii) easy-to-use, as well as (iii) real-time capable approach that comprises pressure sensor on the bow of the violin player. The sensors measure the finger pressure during execution of movements of the bow. The method described here is a further development of the approach of Grosshauser [2008]. However, the sensing technologies we used in this study extend this approach by measuring the pressure of every finger.

6.6.1. Data Set

In order to record the pressure levels of each finger of the right hand, we fixated *force sensing resistor* (FSR) foils to the bow's frog, where the fingers contact the bow (Figure 6.7). Thereby, we placed one FSR underneath the thumb, forefinger, little finger, and another single FSR under ring and middle finger. In addition, we used a goniometer to measure the direction and range as well as the rotation of the bow's motion.

Our self-made goniometer is equipped with a potentiometer and is used for joint angle measurement. The goniometers are fixated at the right hand elbow and wrist. Figure 6.8

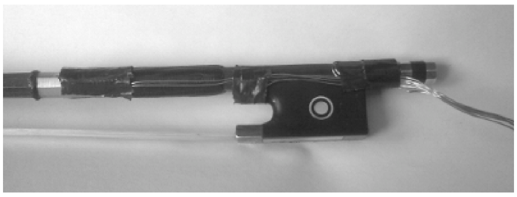


Figure 6.7.: This picture shows the pressure sensors that are attached to the bow. The picture was taken after recording approximately 3000 bowing examples.

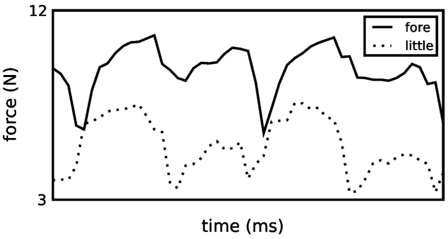


Figure 6.8.: This plot illustrates the development of fore- and little finger's pressure during Martele up- and down-bow strokes.

shows an example of the recorded pressure values changing in time.

For evaluating our setup, we recorded (i) different up- and down bowings (Martele, Detache and Spiccato) (ii) in different tempi, (iii) and defined inaccurate bowings in terms of traditional violin playing. The inaccurate examples were (i) wrong angle with more or less than 90° between bow and the strings, and (ii) too much or not enough pressure in combination with too long or short pressure. In addition, (iii) wrong pressure distribution and inaccurate pressure changes before and during the stroke were executed. We recorded each of these bowings around 200 times (cf. Table 6.11). Figure 6.8 shows a typical measurement of a Martele stroke, i.e., the development of the fore- and little finger during four strokes.

We used data recorded from a professional violinist. The violinist participated in several recording sessions in our laboratory during one week. The recording setup was simple: the sensors were attached to the bow as described in the previous section. The data were recorded by a standard laptop computer connected to the sensor board via Bluetooth. The required playing techniques were presented to the subject by instruction and in musical notation.

In order to prepare the captured data for classification, we used the sensor data from the goniometer for stroke detection. We partitioned the continuous data stream captured by the goniometer into fragments, each ranging from a local maximum to the nearest subsequent local minimum and vice versa. We only used the corresponding data fragments of the pressure sensors for the classification experiments. Please note that we did not distinguish between up-bows and down-bows in this scenario. Table 6.11 gives a brief overview of the data collected in this process.

According to the collected data, we established various classification tasks with different class sets:

(A) **"bowings slow":** Martele vs. Detache vs. Spiccato with slow stroke frequency (≈ 1Hz),

(B) **"bowings fast":** Martele vs. Detache vs. Spiccato with fast stroke frequency (≈ 2Hz),

(C) **"detache angle":** correct Detache strokes vs. incorrect Detache strokes with an angle lower then 90° vs. incorrect Detache strokes with an angle above 90°,

identifier	run	number of examples
correct executions:		
MCS	martele slow	206
MCF	martele fast	198
DCS	detache slow	196
DCF	detache fast	156
SCS	spiccato slow	207
SCF	spiccato fast	192
inaccurate executions:		
MIA>90	martele angle >90	170
MIA<90	martele angle <90	203
MIP+	martele pressure+	133
MIP−	martele pressure−	206
DIA>90	detache angle >90	214
DIA<90	detache angle <90	206

Table 6.11.: This table shows the number of examples per recorded condition.

(D) "martele angle": correct Martele strokes vs. incorrect Martele strokes with an angle lower then 90° vs. incorrect Martele strokes with an angle above 90°,

(E) "martele pressure": correct Martele strokes vs. incorrect Martele strokes with to much pressure vs. Martele strokes with not enough pressure.

The class sets (A) and (B) are concerned to general separability of bowings, whereas class sets (C), (D), and (E) are related to the question of detecting inaccurate bowing executions.

6.6.2. Experiments

In order to evaluate the proposed system, we chose an experimental setup for investigation of the following questions:

1. Can the data from the pressure sensors be used to classify bowings and if so, to what accuracy?
2. Is it possible to detect inaccurate bowing executions by the data form the pressure sensors?

In order to answer these research questions, we conducted the following experiments. We applied a standard 5-fold cross validation procedure on the training data to obtain optimal OMM hyperparameters (K, σ) (c.f. Section 5.2), and then used the hyperparameters found in this process to test the generalization properties of the classifier by means of the remaining testing data. In this process, we chose equal values for the number of OMM states K and the global standard deviation σ in all experiments with $K \in \{2, 4, \ldots, 20\}$ and $\sigma \in \{2.0, 1.6, 1.28, 1.02, 0.81, 0.65, 0.52, 0.41, 0.33, 0.26\}$. We repeated this process 100 times for each class set to enable simple statistics on the achieved error rates.

class sets	classes	error rate	
		mean	std
(A)	MCF, DCF, SCF	0.02	0.01
(B)	MCS, DCS, SCS	0.03	0.01
(C)	DCF, DIA>90, DIA<90	0.13	0.02
(D)	MCF, MIA>90, MIA<90	0.09	0.02
(E)	MCF, MIA+, MIA−	0.02	0.01

Table 6.12.: Error rates from the experiments for different class sets: (A) "bowings slow", (B) "bowings fast", (C) "detache angle", (D) "martele angle", (E) "martele pressure".

6.6.3. Results and Discussion

The results of the bowings classification experiments are summarized in Table 6.12 and Figure 6.9. As a main result, OMMs are able to correctly classify examples from all class sets with high accuracy. For the 100 runs of class sets (A) and (B), which are related to bowing detections, we found a mean error rate of $\approx 2\%$ and $\approx 3\%$, respectively, with a standard deviation of $\approx 1\%$ in both cases. OMMs provide very stable recognition rates, and there seems to be almost no dependency on the execution speed.

Additionally, OMMs detect incorrect bowing execution with reasonable accuracy. For the class sets (C) and (D), which are related to incorrect bowing angles, the classifiers reached a mean error rate of $\approx 13\%$ and $\approx 9\%$ with a standard deviation of $\approx 9\%$ and $\approx 2\%$, respectively. On the other side, the classifiers achieved an error rate of $\approx 2\%$ for detection of incorrect bowing pressure in class set (E). This indicates that identification of inaccurate bowings depends on the type of the inaccuracy. However, even the highest error rate of all 500 runs is below 20% (see Figure 6.9) and provides very good assumptions about if and what mistake occurs.

With regards to this thesis' major research questions, the given results indicate that OMM-based classifiers provide a sufficient representation to recognize bowing types and incorrect bowings. These finding further fortify the results from the previous experiments related to classification: OMMs are able to recognize interaction data with high accuracy **(Q1)**.

6.7. Incremental Recognition and Adaptive Learning

In this Section, we will evaluate an incremental classification approach where a classification decision is updated while an behavior is observed. In addition, we will evaluate an adaptive, online training scheme that updates an OMM observation by observation, as they arrive. Therefore, we will examine an interaction scenario where a human plays a variant of the well-known rock-paper-scissors game against the virtual agent VINCE. The proposed classification scheme for incremental classification is presented in-depth in Section 6.1.1, while we introduce our adaptive online training approach in Section 6.1.2.

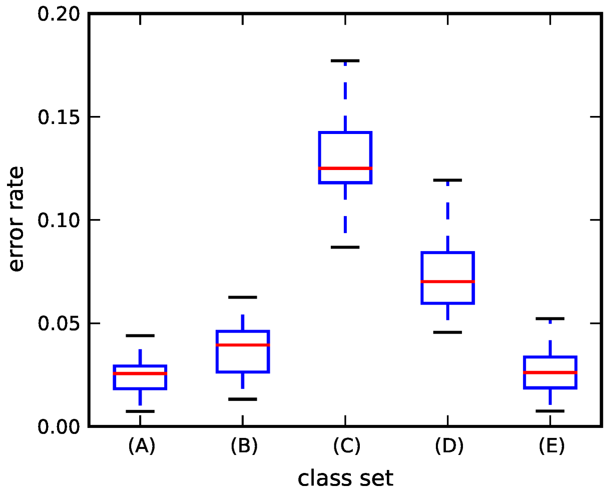


Figure 6.9.: Box plots of the error rates for each class sets from 100 runs. The x-axis denotes the error rate and the y-axis the class set (A) “bowings slow”, (B) “bowings fast”, (C) “detache angle”, (D) “martele angle”, (E) “martele pressure”

6.7.1. Data Set

In order to evaluate the proposed approach and to demonstrate its abilities for incremental recognition and adaptive learning, we implemented an interaction in which fast adaption and quick online recognition are crucial and have observable consequences for the interaction. Specifically, we realized an extended version of the rock-paper-scissors (RPS) game for one human player and the virtual agent VINCE.

In its classic form, RPS is a hand gesture game in which both players simultaneously choose one of three hand gestures: *rock*, *paper*, or *scissors*. To ensure simultaneity (and thus fairness), both players entrain by performing an initial counting-in while saying “Ro! Cham! Beau!” in correspondence to rhythmical arm movement. On ”Beau!”, both players present their hand gestures. The winner of the game is selected by comparison of the gestures. In the classic variant with three gestures *rock* beats *scissors*, *scissors* beats *paper*, and *paper* beats *rock*, which leads to balanced chances for the gesture selection. RPS exists in variation since ≈ 4000 years, and since 2002 the *World Rock Paper Scissors Society* organizes an annual world cup.

We adopted a famous extension to RPS that adds two extra gestures to the game, a **lizard** and a **spock** gesture. We will refer to it as *rock-paper-scissors-lizard-spock*, RPSLS. Figure 6.10 shows a resolution diagram for all gesture combinations of RPSLS. In this variant each gesture wins against two gestures and looses against the remaining two gestures, again, balancing the chances to win. The figure also shows the corresponding gestures we used in our implementation of the game (Section 6.7.1 gives the rationale for using them).

In our scenario, we enabled VINCE to play RPSLS, but based on abilities for incremental behavior coordination: the agent does the initial counting phase in sync with the player, but instead of performing a pre-chosen gesture on ”Beau!”, VINCE tries to recognizes as rapidly as possible the gesture of the human player and then performs a corresponding

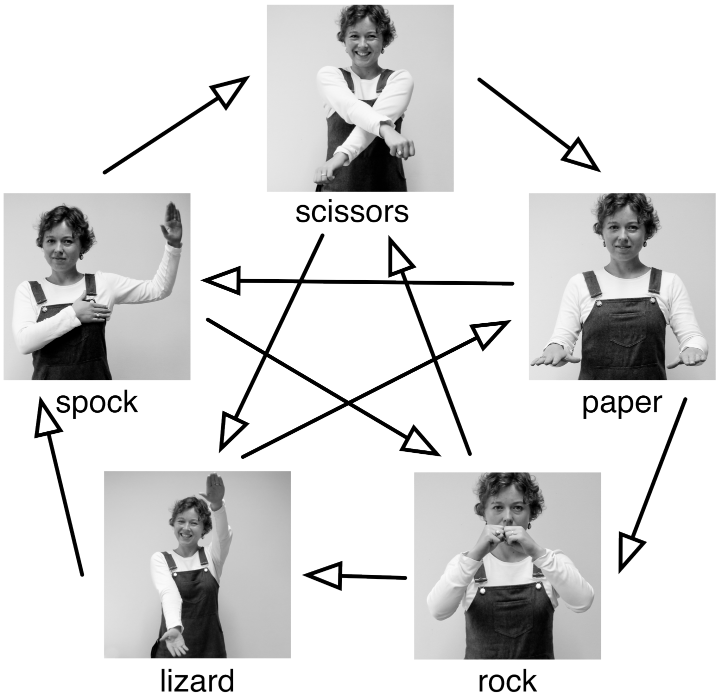


Figure 6.10.: Gestures included in the *rock-paper-scissors-lizard-spock* game along with the winning relations.

winning gesture himself. The agent thereby learns and adapts to the particular user-specific way of performing a particular hand-arm gesture via online training of the models. Further, VINCE uses the learned OMM prototype models to realize gestures during the game himself, thus reproducing the observed behaviors and coordinating with the user. Please see Chapter 7 for a detailed introduction to reproduction of interaction data with OMMs.

Setup

In our setup shown in Figure 6.11, a human player competes against the virtual agent VINCE. A Microsoft Kinect^{TM} sensor captures the scene in 3D at a frequency of $\approx$ 30 fps. The scene data are processed using the OpenNI[5] framework to extract a human skeleton for the present user in the scene. We captured the relevant information of the skeleton, such as the user's height, spatial positions of the wrists and the center of mass to compute the normalized 3-dimensional positions of the wrists with respect to the user's body center. In addition, we capture the user's right hand finger configuration by means of an Immersion CyberGlove device. Here, we recorded 24 joint-angles of all five fingers, again at a frequency of $\approx$ 30 fps. Note that RPS (as well as RPSLS) is classically played with different hand postures. We have successfully realized this variant with the CyberGlove data as input vectors for the OMMs. We chose to only use the data captured by the relatively less reliable Microsoft Kinect, since initial pre-study experiments showed that recognizing arm gestures is the harder and hence more revealing task compared to hand gestures captured

[5] http://www.openni.org

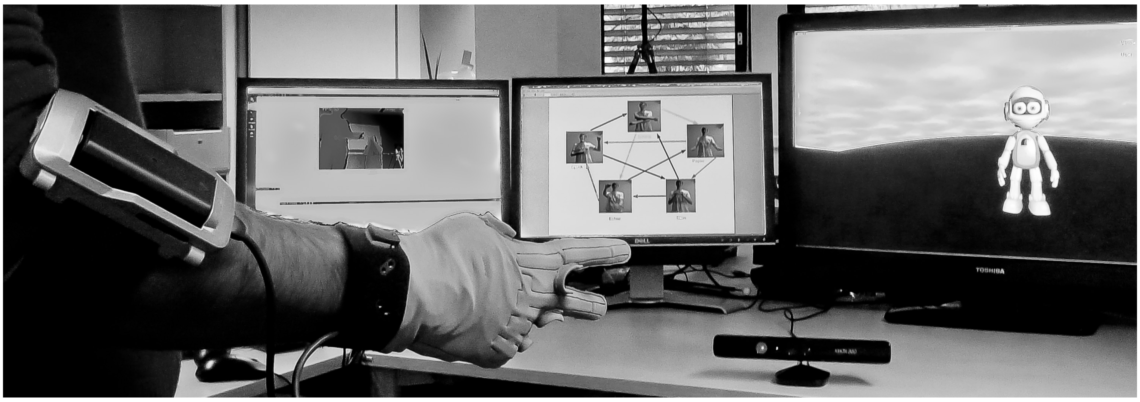


Figure 6.11.: Setup of the RPSLS game; the Microsoft Kinect^{TM} sensor is on a table in front of a monitor on which the virtual agent VINCE is displayed. The leftmost monitor displays the software that captures the Kinect data, the monitor in the middle shows the gestures as a reminder for the player. The arm of the human player wearing a CyberGolve device is in the front.

by the CyperGove. Therefore, we replaced the classical hand postures with gestures to be executed with both arms. Thus, the player presents the chosen gesture and configuration of both arms after performing the (unchanged) counting-in step. See Figure 6.10 for pictures of the arm configurations used in our scenario.

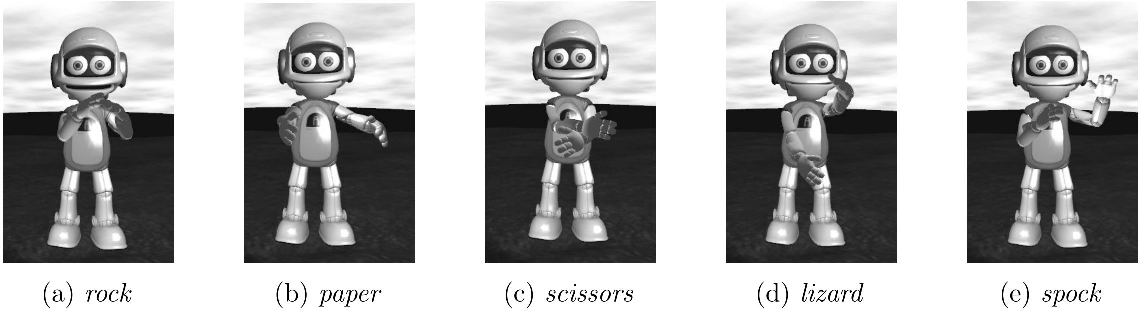
As additional communication channel during the game, the player can give verbal commands to the system via a head mounted microphone. For speech recognition, we use the Microsoft Windows 7 Speech Recognizer. VINCE, in turn, generates synthetic speech and lip sync speech animation using the Articulated Communicator Engine [ACE, Kopp and Wachsmuth, 2004].

The course of the game is as follows:

1. VINCE welcomes the player and asks to start a game
2. VINCE offers to demonstrate all five gestures
3. VINCE adopts a starting position and asks if the human player is ready
4. If agreed, the next round of the game starts.
5. VINCE, in sync with the human player, executes the *counting in* arm movements while saying "Ro! Cham! Beau!" with a time lag of 510ms between each word.
6. Both players perform a gesture on (or shortly after) "Beau!".
7. VINCE indicates which gesture he thinks the player did, and then asks which gesture the player has actually performed
8. The game score of the player with the winning gesture increases by 1.
9. When one player reaches a score of 20, the game ends. Otherwise, another turn begins at 3.

That is, a game takes at least 20 rounds during which VINCE learns and adapts to the player's gestures in an, at least, partly supervised manner.

The agent recognizes the gestures the user performs by means of an $\text{OMM}_{\text{incremental}}$-based classifier. This classifier returns a classification decision if the likelihood ratio between the most-likely and the second-most-likely OMM exceeded a value of 2 or, latest, $310ms$



(a) *rock* (b) *paper* (c) *scissors* (d) *lizard* (e) *spock*

Figure 6.12.: Screenshots of VINCE performing the learned OMM prototypes for the five classes.

after "Beau!". Therefore, the user gets the impression that VINCE presents his gesture without noticeable delay. Depending on the outcome of the recognition, the agent chooses a gesture that will win against the one picked by the player. The game begins with an unlearned classifier that adapts to the gestures performed by the user. After each turn, the classifier is re-trained with the new observation according to the adaptive training scheme as presented in Section 6.1.2. Thus, in the beginning, the human or the agent will win by chance, but VINCE's abilities to predict the gestures of the player improve rapidly during the course of the game. In order to keep the human player motivated and to increase the amount of example data, we let VINCE randomly perform a loosing gesture if VINCE is 5 points in the lead.

In our setup, the agent VINCE adapts to the gestures the human player performs. The adapted models enable better prediction and classification, but are also used when VINCE performs gestures himself. VINCE uses the OMM prototypes that the system learns in the course of the game as his own gesture model. Therefore, VINCE re-produces the learned mean values. Adaptation thus leads to increased reproduction and imitation. Since the OMMs emit mean 3D wrist locations in working space, the coordinates of the user's wrists positions are transformed from the coordinate system of the player's egocentric location space into the virtual agent's body-centered frame of reference. Using inverse kinematics, VINCE can then reproduce the human's gesture in concordance with his body relations. Before a model is available, i.e., before the user presented a gesture to VINCE, we use pre-recorded gesture trajectories. Please see Chapter 7 for details about reproduction of interaction data with OMMs.

Based on the demonstration setup, we conducted an evaluation study with 11 participants who played the game with VINCE until either player reached a score of 20. We recorded the wrist positions of both arms as location coordinates relative to the user's body center for later analysis. The recording started when VINCE executes the second count ("Chamb") and ends 1110 milliseconds later. Table 6.13 gives an overview about the collected data and the participants. Overall, we collected a data set containing 408 gestures in five classes. Figure 6.12 shows screenshots of VINCE performing the learned OMM prototypes as gestures during the game, i.e., after a few rounds. The obvious correspondence to the users' gestures, as displayed in Figure 6.10, demonstrates the ability of the approach to rapidly acquire novel gestures and to learn models that enable the agent to perform the (prototype of the) behavior itself.

player	age	gender	rock	scissors	paper	Spock	lizard	$\sum$
1	27	male	7	7	8	6	5	33
2	24	male	5	8	10	7	5	35
3	23	female	11	7	6	4	5	33
5	20	male	14	7	10	8	7	46
6	37	male	6	5	9	7	8	35
7	20	male	12	12	10	8	5	47
8	21	male	10	8	9	9	10	46
9	24	female	7	5	9	5	4	30
10	24	male	9	8	8	9	8	42
11	24	male	6	7	6	8	6	33
12	23	female	4	6	5	7	6	28
$\sum$			91	80	90	78	69	408

Table 6.13.: Here, age and gender of the human players as well as the number of gestures recorded in the study are shown.

6.7.2. Experiments

We analyzed the collected data set according to the following questions:

1. What influence does the choice of $\text{OMM}_{\text{incremental}}$ hyperparameters have on the recognition performance?
2. What recognition accuracy do $\text{OMM}_{\text{incremental}}$ classifiers reach compared with standard classifiers in sample-by-sample classification?
3. Are $\text{OMM}_{\text{incremental}}$ classifiers able to adapt to the user over the course of a game?

To address the first question, we evaluated a set of OMM hyperparameter pairs by means of 10-fold cross validation on the complete data set. Here, we varied the number of OMM states $K \in \{3, 5, 8, 10, 15, 20, 25, 35, 40, 50, 60, 80\}$ and standard deviation parameter of the emission densities $\sigma \in \{0.1, 0.2, \ldots, 1.1\}$ as hyperparameters. In order to address the second question, we compared the $\text{OMM}_{\text{incremental}}$ classifiers to a widely used classification technique for time series, a Nearest Neighbor Classifier with Dynamic Time Warping as a distance function (NN_{DTW}, please see Section 3.3.2 and Section 3.2.1 for in-depth introductions to these techniques). We applied a uniform procedure to both classifiers and evaluated the recognition accuracy in the course of a gesture by means of a 5-fold cross validation. We applied a sample-based $\text{OMM}_{\text{incremental}}$ procedure to the fold's testing data. To simulate an incremental approach with NN_{DTW} classifiers, we used the corresponding sub-time series from the beginning to the current point t in time. We identified optimal OMM hyperparameter by means of a 5-fold cross validation on the training data beforehand. To address the third question, we evaluated the development of the classification accuracy during the course of the game for all 11 participants.

	Lizard	Paper	Scissors	Spock	Rock
Lizard	20	0	0	1	0
Paper	0	29	2	1	3
Scissors	0	1	26	0	1
Spock	1	0	0	28	2
Rock	1	0	1	0	33

Table 6.14.: Confusion matrix from the classification experiments. The columns are the actual labels and the rows the classification decision. The cells of this table indicate how many elements of a genuine class are classified as all classes.

6.7.3. Results and Discussion

Figure 6.13 shows the cross validation error rate for our gesture data set, depending on the OMM hyperparameters (K, σ). As can be seen, the classification accuracy is stable for the evaluated number of model states K and the standard deviation parameter σ. On average, $\text{OMM}_{\text{incremental}}$ classifiers yield a classification error of ≈ 0.15 with a standard deviation of ≈ 0.04. The lowest error rate of ≈ 0.08 yields a classifier that was trained with $K = 40$ and $\sigma = 0.2$. A model trained with only $K = 5$ model states and $\sigma = 0.7$, however, still reached a cross validation error of only ≈ 0.13. These results show that $\text{OMM}_{\text{incremental}}$ classifiers are able to recognize behavior patterns with high accuracy, even with not optimally adjusted hyperparameter values. This neat property allows to use $\text{OMM}_{\text{incremental}}$ classifiers with a smaller number of model states, e.g., in scenarios with little computational resources or severe time constraints, while still reaching good recognition results.

Figure 6.14 shows the development of the production likelihoods of the $\text{OMM}_{\text{incremental}}$ models for the five different gestures, during observing an example gesture of class *spock*. In the beginning, all models are equally likely to produce that particular gesture in question. However, after $\approx 600ms$ (approximately on "Beau!") the model related to class *spock* stably stays on a likelihood level of $\approx 10^{-13}$ while the likelihood associated with the other models decreases to a minimum of $\approx 10^{-79}$. In this case, a recognition of the gesture is possible $\approx 600ms$ after the gesture performance begins, i.e., in synchronization with the gesture presentation.

Figure 6.15 gives the results of the comparison of the recognition errors of $\text{OMM}_{\text{incremental}}$ and NN_{DTW} classifiers in online classification. As can be seen, the recognition accuracy of both classifiers increases with each additional sample becoming available. For complete gesture performances, NN_{DTW} classifiers reach a slightly lower error rate of ≈ 0.12 in contrast to a recognition rate of ≈ 0.18 for $\text{OMM}_{\text{incremental}}$ classifiers. However, for partial gesture performances, $\text{OMM}_{\text{incremental}}$ classifiers yield up to $\approx 10\%$ (on average $\approx 5\%$) lower error rates. These results indicate that $\text{OMM}_{\text{incremental}}$ classifiers are well suited when incremental recognition of behavior patterns is required.

Last, Figure 6.16 shows the average learning curves (averaged over all five gestures) for all 11 games. In the beginning, VINCE is unable to recognize a gesture the human player performs, because the OMMs are neither learned nor adapted to the specific user (recognition error rate of 1.0 for the first turn). The decrease, however, especially over the first 10-15 rounds, demonstrates the rapid learning ability of the used $\text{OMM}_{\text{incremental}}$

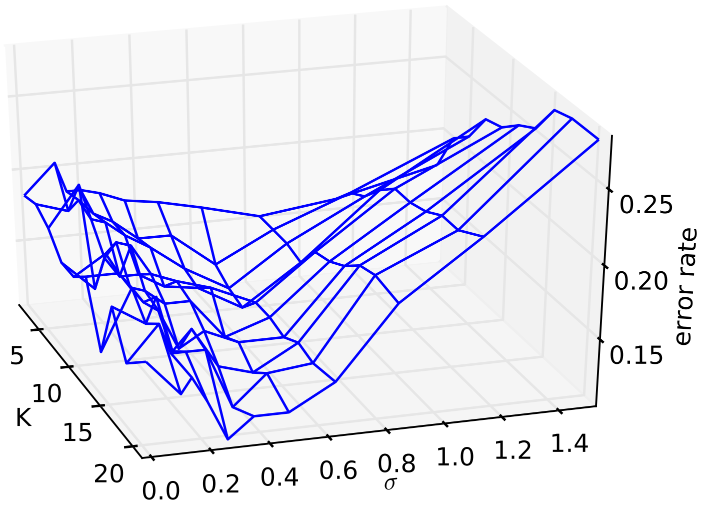


Figure 6.13.: This figure illustrates the achieved classification accuracy (in cross validation) for different numbers of OMM states K and different standard deviation parameters σ.

classifiers. The error rate decreases down to an average recognition error of ≈ 0.15. The characteristic drops occur when the user performs a gesture for the first time, or in a very different way than before. In total, VINCE managed to win all 11 games with a lead of at least 3 points.

6.8. Summary

The aim of this chapter was to investigate if OMMs are capable to recognize interaction data that is modeled as time series. We investigated the general applicability of OMMs to recognition of interaction data with a focus on response times and OMMs' robustness properties. Further, we examined an incremental recognition approach, as well as an adaptive, online learning scheme to reflect changing behaviors.

In **Section 6.1**, we presented the theoretical outline of this chapter. Here, we first described a maximum-likelihood approach to classification with OMMs, followed by an extension to incremental classification of time series that allows on-the-fly recognition of behavior patterns while they are observed (Section 6.1.1). The section closes with a description of an adaptive online learning scheme (Section 6.1.2).

In the subsequent **Section 6.2**, we evaluated if OMM-based classifiers are able to recognize interaction data at all. We examined the classification performance of OMM-based classifiers in comparison to standard approaches for time series classification: Nearest Neighbor Classifiers with Dynamic Time Warping distance function and Support Vector Machines with Gaussian Dynamic Time Warping kernels. The results from experiments on real-world

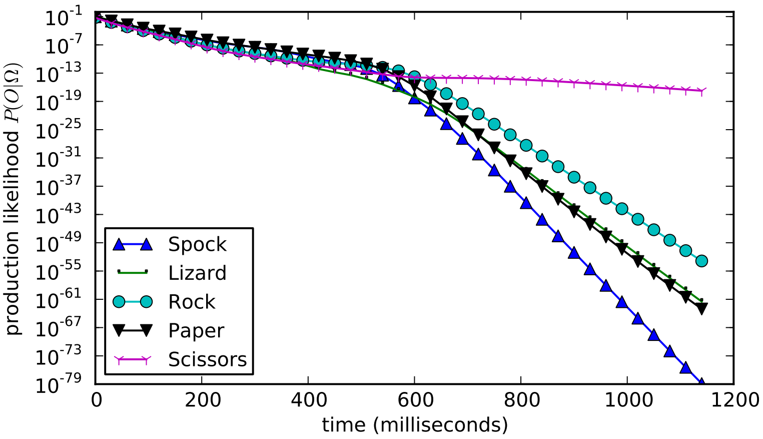


Figure 6.14.: Likelihoods of class-related OMM$_{\text{incremental}}$ models during observation of a *spock* gesture. Note how the likelihood difference between an OMM$_{\text{incremental}}$ that corresponds to *lizard* and the remaining OMM$_{\text{incremental}}$ substantially increases after $\approx 600ms$.

interaction time series data sets indicate that OMMs are well suited to process interaction data; and in contrast to highly accurate kernel machines, the computational efficiency of the OMM classifiers is substantially low and, thus, allows real-time recognition in various scenarios.

In **Section 6.3** we focused on the robustness properties of OMMs in terms of incomplete observations. Therefore, we compared OMM-based classifiers with a topologically similar HMM-based approach. In order to investigate the robustness of the proposed models, we designed a simulation experiment where we evaluated the classifiers with fragmented test data of decreasing length. We found that the error rates of HMMs increased dramatically on time series fragments, whereas OMMs still performed well up to a fragment length of 20% of the genuine time series lengths. These results indicate that the omission of transition probabilities, among enormously reduced computational demands, has a positive impact on the robustness with respect to fragmented time series observations.

To further underline these findings, we subsequently examined the influence of transition probabilities by means of 20 benchmark data sets that are related to various time series domains in **Section 6.4**. We evaluated OMMs as well as HMMs with various topologies (HMM$_{\text{LR}}$, HMM$_{\text{LIN}}$, and HMM$_{\text{PDTP}}$) with full and artificially fragmented testing examples of half-length. The results indicate that the simplification provided by OMMs significantly increases the robustness with regard to classification of incomplete data as compared to HMMs. Although OMMs provide higher model flexibility and consistently superior classification accuracy, the computational cost is even lower than that of linear HMMs. In comparison with equally flexible left-to-right HMMs, the considerable speed-up is practically relevant for interaction scenarios. Therefore, the duration modeling that arises from the omission of transition probabilities represents a valuable alternative to standard HMMs and their various extensions.

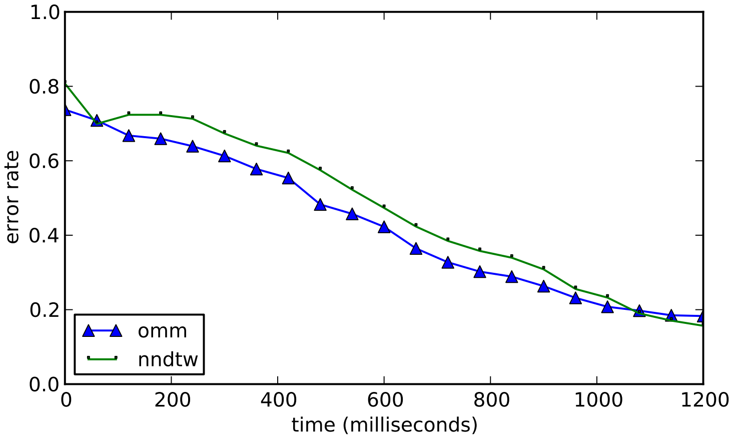
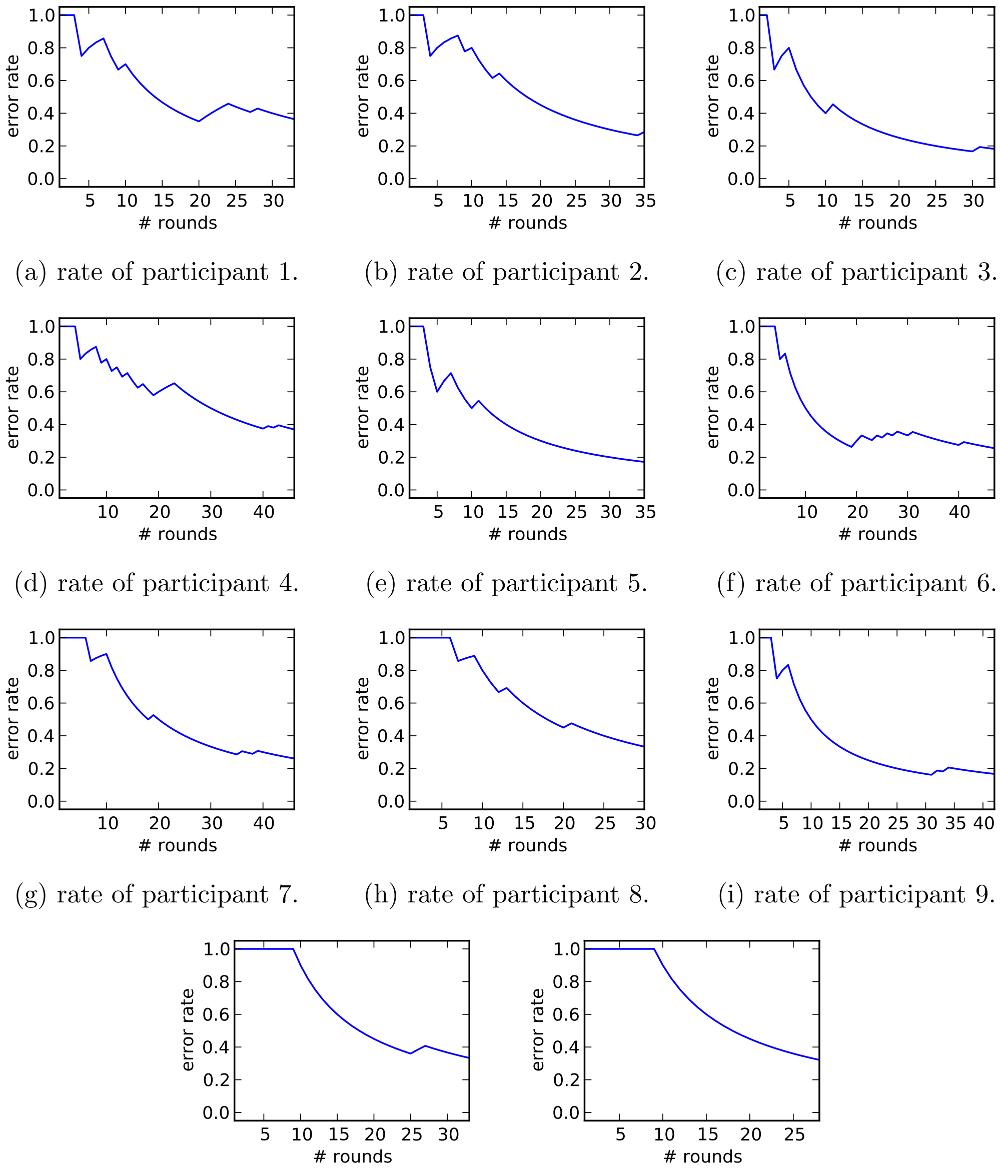


Figure 6.15.: Change of recognition accuracies of $\mathrm{OMM}_{\mathrm{incremental}}$ and $\mathrm{NN}_{\mathrm{DTW}}$ classifiers over time (during observation of gestures).

In **Section 6.5** we introduced and evaluated an interaction scenario related to head gesture recognition. We conducted a comprehensive study with 10 subjects, where we captured head gesture examples by means of 3 DoF goniometer recordings. Subsequently, we analyzed the emerging data set that contains four head gestures classes by means of OMMs with regard to separability and incomplete observations. The results of this experiment indicate that OMM-based classifiers are able to represent head gesture data with high accuracy. However, the classification performance depends heavily on the chosen classes: class sets that include *shake* and *look* classes yield substantially more mis-classifications. In terms of incomplete observations, OMMs are able to recognize head gestures with comparatively low error rates even if only 10% of the genuine head gesture is performed, what indicates that head gestures are still well separable if only partial observations are available. This leads to the conclusion, that OMMs are able to provide a reasonable hypothesis after a short observation period.

In **Section 6.6** we explored an application driven scenario for recognition of bowing types and incorrect bowing executions for violinists. In this setup, we measured the pressure of each finger and the thumb of the right hand (the hand that holds the bow) by means of Force Sensing Resistors. Subsequently, we applied OMM-based classifiers to detect (i) which bowing type is performed (Martele, Detache, Spiccato) in concurrency with different timings, (ii) if a bowing is played incorrectly, e.g., with too much or not enough pressure, or with an incorrect angle between bow and strings. The results of this study showed that OMMs-based classifiers are a well suited choice to detect bowing types and mistakes in bowing handling. The combination of single finger pressure measurements and OMMs is a promising approach to improve violin teaching and exercises.

The aim of **Section 6.7** was to evaluate adaptive online learning and the incremental classification scheme that allows classification while an observation is made and before all samples are present. To see whether OMMs work in such a way, we realized the Rock-Paper-Scissors game as an application and test scenario in which the virtual agent VINCE engages in fast, real-life gesture-based interaction with human users. This scenario directly



(a) rate of participant 1.

(b) rate of participant 2.

(c) rate of participant 3.

(d) rate of participant 4.

(e) rate of participant 5.

(f) rate of participant 6.

(g) rate of participant 7.

(h) rate of participant 8.

(i) rate of participant 9.

(j) rate of participant 10.

(k) rate of participant 11.

Figure 6.16.: These plot illustrate the average learning curves (averaged over all five gestures) for all 11 human player's games.

taps into the abilities of the model for incremental recognition and adaption. The results that we obtained in an evaluation study indicate that OMMs can meet all before-mentioned requirements: rapid, incremental recognition with high accuracies; as well as the ability to adapt to a particular user via online re-training of the classifiers.

In terms of the **major research questions** that were defined in Section 2.1, the given experimental results indicate that: (i) OMM classifiers are able to recognize interaction and behavior time series data with an accuracy that is comparable to standard classification approaches of time series **(Q1)**. (ii) OMMs allow low latency classification, which makes them particularly well suited for interaction scenarios where rapid response times are crucial **(Q2)**. (iii) In addition, OMM-based classifiers perform outstanding in terms of incomplete observations as compared to similar HMM-based classifiers. Here, the absence of transition probabilities leads to a flexible modeling of the observation lengths distribution. This might be the reason why OMMs are superior in recognizing fragmented observations, which are, in principle, observations from a known class but drawn from a different lengths distribution **(Q3)**. (iv) The incremental classification scheme of OMMs allows updating classification decisions while behaviors are observed **(Q5)**. (v) The online re-training of OMM models supports successful adaption to particular users and environments **(Q4)**.

7. Reproduction of Interaction Time Series Data with OMMs

As stated in Chapter 1, reproduction of user interaction is of utmost importance for many interaction scenarios and applications. I.e., in human-human interactions we find that gestures between communicating people are tightly interweaved and thereby successfully support and structure interaction. Obviously, the interactants make sense of their observations incrementally and can foresee the continuation and/or intervene and react gutturally without significant delay. Particularly imitation is a behavior pattern observed frequently in human-human interaction where it supports and establishes communicative alignment. However, beyond communicative functions, gesture reproduction is also needed if machines are to learn behavior patterns from example, thus, allowing commanding future robots or agents by just demonstration.

In this chapter, we analyze the capabilities of OMMs for gesture reproduction. The experiments are directly related to the following major research questions of this thesis:

- **(Q6) Prototype Property:** Do OMMs provide interpretable prototypes of interaction data, i.e., to what degree is the learned array of reference vectors a reliable prototype representation for a given data set?
- **(Q7) Influence of Hyperparameters:** What influence does the choice of the OMM hyperparameters, namely, the deviation parameter σ and the parameter that indicates the number of model states K have?
- **(Q8) Training Process:** To what degree is the development of the learned OMM representations related to the development of the objective function?

This chapter is structured as follows: After addressing the term prototype as it is used in this thesis in Section 7.1, we investigate if OMMs are able to reproduce interaction data accurately. We initially use OMMs to analyze computer mouse gestures in Section 7.2. Section 7.3 is related to an application in human-agent interaction that incorporates imitation learning of natural hand-arm gestures by the artificial agent VINCE.

7.1. Definition: Prototype

Mathematical modeling of quantitative observation is a common technique in statistics, machine learning, data mining, and numerous other scientific and engineering disciplines. The term *model* usually characterizes a (simplified) description of a system. Sometimes, e.g., in case of machine learning and data mining, a model describes a given set of observations, effects, or events. For example, statistical models such as random variables incorporate certain distributions that are often used to qualify observed data that share a distinctive quality of a particular type. Such a model usually characterizes the data with few parameters, e.g., in case of multivariate Gaussian distributions, the data is represented by an expectation value vector $\boldsymbol{\mu}$ and a covariance matrix Σ:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left\{-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}. \tag{7.1}$$

$$\text{with } \mathbf{x}, \boldsymbol{\mu} \in \mathbb{R}^d,\ \Sigma \in \mathcal{M}_{\mathbb{R}}\,(d,d)\,. \tag{7.2}$$

Here, $\mathcal{M}_{\mathbb{R}}\,(d,d)$ denotes the matrix space of size $d \times d$.

Such a model may help to explain a system and to study the influence of various components, or to make predictions about the model's future behavior.

We understand the term prototype in this thesis as a typical representative for a given set of observations. The Oxford Dictionary of English [Pearsall and Hanks, 1998] defines *prototype* as

> *the first, original, or typical form of something; an archetype."*

Such a definition induces that a prototype is of the same type as the observations. In contrast to a model, a required prior condition for prototypes is that they could be elements of the genuine set of observations.

By this definition, the above-mentioned Gaussian distributed random variable (Equation 7.1) is not a prototype. Nevertheless, since the expectation vector $\boldsymbol{\mu} \in \mathbb{R}^d$ of the Gaussian distribution is an element of the same vector space as the observation $\mathbf{x} \in \mathbb{R}^d$. Thus, the ordered sequence of expectation vectors $\Omega = \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_T$ defines a times series that can be used as a prototype for the observed times series.

In these terms, Ordered Means Models provide a prototype as well as a mathematical model for time series observations. This is a unique property that we do not find in other time series models.

7.2. Mouse Gestures

Mouse gestures are an alternative modality in human computer interaction and widely used in, e.g., mobile devices or web browsers. Even though many applications already exist that incorporate mouse gestures[1], and mouse gesture recognition is not a challenging task anymore, the here used mouse gesture data set offers us the opportunity to investigate and illustrate basic research questions related to reproduction and prototyping.

7.2.1. Data Set

To answer the above-mentioned research questions, we designed a simple research study. The study is situated in a common office place, with the participants sitting in front of a desk that holds a 13" computer monitor and a computer mouse. Figure 7.1 illustrates the experimental setup. The monitor shows a plain white screen and a common mouse pointer that is shaped like a small arrow. After a short introduction about the aim of the study, we ask the participants to perform three gestures with the computer mouse:

[1] one of the first widely available applications that incorporated mouse gestures was the opera web browser `http://www.opera.com/browser/tutorials/gestures/`. Opera introduced mouse gesture in April 2001.

Figure 7.1.: This figure illustrates the setting of the mouse gesture experiment. The scenario is a standard office scenario.

- **Spiral:** a counterclockwise spiral that unfolds from the center,
- **Circle:** a clockwise circle gesture,
- **Figure Eight:** a horizontal figure eight also known as the infinity symbol.

Initially, the supervisor demonstrates the mouse gestures once. The emerging mouse gesture trajectories that are performed by the participants are recorded as 2-dimensional location coordinates with time stamps. Eleven human demonstrators participated in this study, all of whom were computer science students and well-trained computer and computer mouse user. Mouse gestures can be plotted as 2-dimensional spatial diagrams and, therefore, can be easily inspected visually.

The collected mouse gestures data set was recorded at 60Hz and contains 33 trajectories in three classes. The average length of the mouse gesture trajectories is 1.6 seconds (within a standard deviation of 0.64 seconds), i.e., ≈100 samples (a deviation of ≈38). The complete data set is available as supplementary material.

7.2.2. Experiments

We used the emerging mouse gestures trajectories to train OMMs — one for each gesture class. We then examined the resulting model parameters according to the above-mentioned research questions.

For comparison, we not only examined OMM prototypes, but also prototypes based on Hidden Markov Models that are obtained in a similar manner as the OMM prototypes: (i) HMMs$_{\mathrm{LR}}$ are topologically similar to OMMs, but incorporate transition probabilities, (ii) HMMs$_{\mathrm{LIN}}$ comprise a linear model topology. Please see Section 5.1 for detailed descriptions of these approaches.

Similar to OMMs, the emission densities of both HMM variants are Gaussian and their standard deviation parameters are used as global hyperparameters. Thus, they both can be partly described as a linear array of reference vectors, i.e., the expectation values of the emission densities. Since these reference vectors are elements of the same vector space as the samples of the genuine time series, the complete array of reference vectors are elements of the gesture space, too. However, given the model design of HMMs, transition probabilities are necessary to completely describe HMMs$_{\text{LR}}$ and HMMs$_{\text{LIN}}$ — an information that is unused in our HMM prototype.

In order to examine the influence of the hyperparameters we repeatedly train OMM- and HMM-prototype representations with different hyperparameter combinations. We use pairs of the deviation parameter σ and the number of model states K according to

$$(\sigma, K) \in \{0, 0.25, 0.5, 1.0, 2.0\} \times \{5, 10, 20, 50, 100\}$$

in this process.

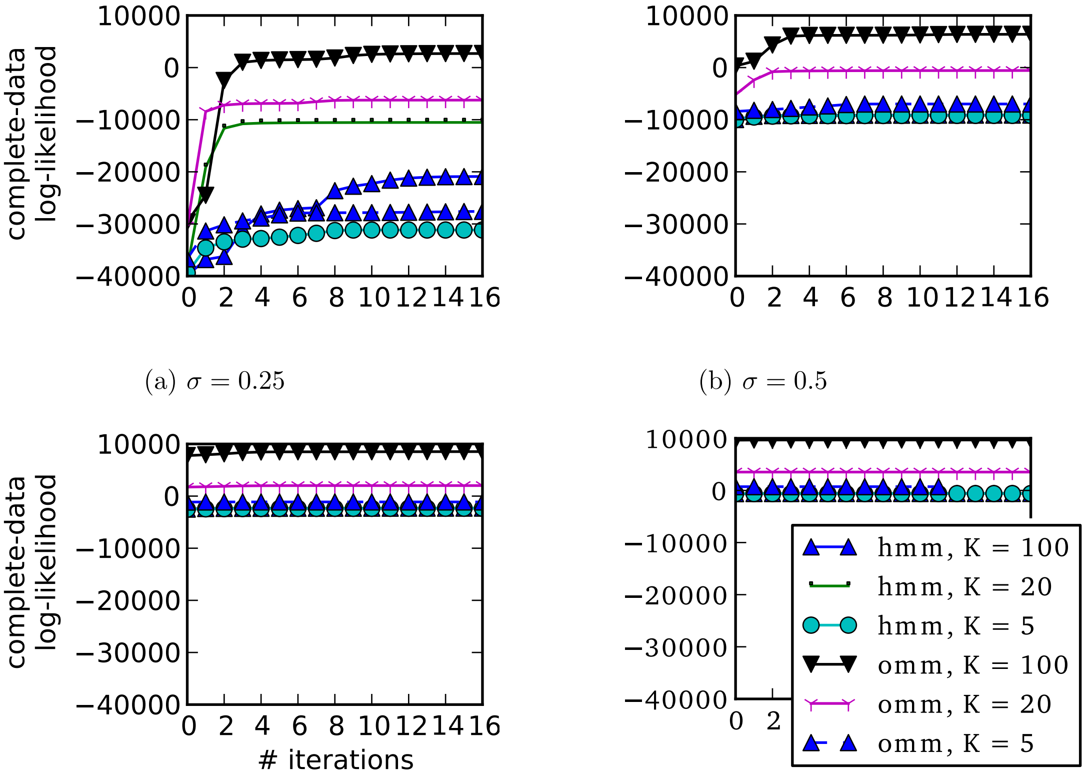
In a similar manner, we examined the influence of the training process on the prototypes. First, we analyzed the course of the complete-data log-likelihood for both OMMs and HMMs. Afterward, we investigated how the mouse gesture models develop during the training process.

7.2.3. Results and Discussion

Figure 7.2 illustrates various developments of the complete-data log-likelihood (i.e., the objective function, cf. 4.10) plotted against the number of iterations. Each sub-figure contains curves for different values of the deviation parameter σ and the number of model states K, as well as a comparison of the evolving log-likelihoods of OMMs and HMMs$_{\text{LR}}$. The plots indicate that the training process of OMMs quickly meets a stable log-likelihood level. For almost all model configurations, the objective function converges after approximately one iteration. Only for $\sigma = 0.25$, the complete-data log-likelihood converges after three iterations, and for $K = 100$, a considerable increase of the complete-data log-likelihood can be observed after nine iterations. In comparison, the objective function of HMMs$_{\text{LR}}$ requires up to 14 iterations to converge ($K = 100, \sigma = 0.25$). However, in general also the complete-data log-likelihood of HMMs$_{\text{LR}}$ quickly converges after only few iteration steps. For higher σ values, the complete-data log-likelihoods do not change substantially after initialization. This might be because σ controls the influence of distant data points in the objective function. In case of $\sigma \geq 1$, also comparatively far distances are fully added to the objective function's outcome. This leads to only minor changes in the value due to training. Summarized, these results indicate that the EM-algorithm of OMMs, as well as the complementary training procedure of HMMs are able to quickly reveal relevant information from a gesture data set.

Figure 7.3 further underlines these findings. The figure illustrates the development of the learned OMM prototype for mouse gestures of class *spiral* depending on the number of iterations with a model size of $K = 100$ and $\sigma = 1$. Similar to the developments of the complete-data log-likelihoods as discussed above, the model does not change substantially after three iterations, with the vastest alternation being between initialization and the first EM-step. In the following iterations, the OMM prototype only alters marginally.

Both results, the analysis of the development of the objective function as well as the



(a) $\sigma = 0.25$

(b) $\sigma = 0.5$

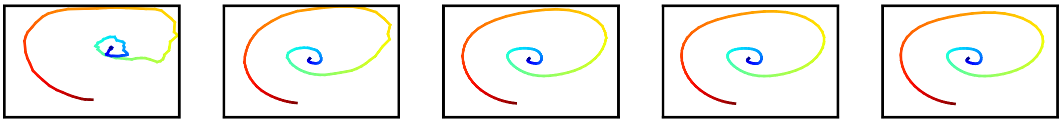
(c) $\sigma = 1.0$

(d) $\sigma = 2.0$

Figure 7.2.: These plots illustrate the development of the objective function for the models used in this study (OMMs, HMM_{LR}) and different values of the hyperparameters. Each plot illustrates the development of the complete-data log-likelihood for a different value of the deviation parameter σ, while combining values for the number of states parameter and the chosen model. Since the results for HMMs_{LIN} are similar to HMMs_{LR}we omit them to increase clarity of the plot.

inspection of the OMM prototype, indicate that the training procedure of OMMs are able to rapidly extract meaningful, characteristic representations from mouse gesture data.

Figure 7.4 illustrates the evolving OMM prototype for different values of the number of model states K and the emission distributions' standard deviation parameter σ exemplary for the mouse gesture class *spiral*. Each row contains plots for fixed values of σ whereas the columns display plots for fixed number of model states K. Theses plots emphasize the effect of the deviation parameter on the model smoothness. For $\sigma \in \{0, 0.25\}$ a trajectory in form of a spiral is, even though evident, very noisy and does not sustain visible inspection. For $\sigma > 0.5$ the visible smoothness considerably increases and unfolds a seemly trajectory in form of a *spiral*. For further increased values of the deviation parameter, also the smoothness increases. E.g., a small dip in the trajectory that can be observed for at top of the plots for values of $\sigma = 0.5$ and $K \geq 50$, dissipated for $\sigma \geq 1.0$. On the other hand, the number of model states K controls the resolution of the learned prototype representation: for models with less than 50 states, single trajectory segments are visible. However, the



(a) iteration #1 (b) iteration #2 (c) iteration #3 (d) iteration #4 (e) iteration #5

Figure 7.3.: These plots illustrate the development of the learned OMM prototype depending on the number of iterations. The first plot shows the model after initialization without any training.

general underlying spiral form can sill be identified. The number of model states has to be chosen according to a desired application.

Figure 7.5 displays OMM prototypes in comparison to HMM_{LR} and HMM_{LIN} prototypes — for all three mouse gesture classes *figure eight*, *spiral*, and *circle*. For all evaluated models, we chose the number of model states as $K = 100$ and the deviation parameter as $\sigma = 1$. The first column shows the prototypes of OMMs, the second and third columns the prototypes of HMM_{LIN} and HMM_{LR}, respectively. The plots indicate that OMMs and HMMs_{LIN} are able to extract meaningful prototypes for mouse gestures trajectories. However, even though the particular trajectory classes can be identified easily for HMM_{LIN} the prototypes plots are more uneven and patchy. In particular, the HMM_{LIN} representation of the *circle* class lacks smoothness in contrast to OMMs' prototype plots that display a seemliness reproduction. The figures clearly indicate that HMM_{LR} are unable to adequately represent the relevant characteristics of the underlying mouse gesture class.

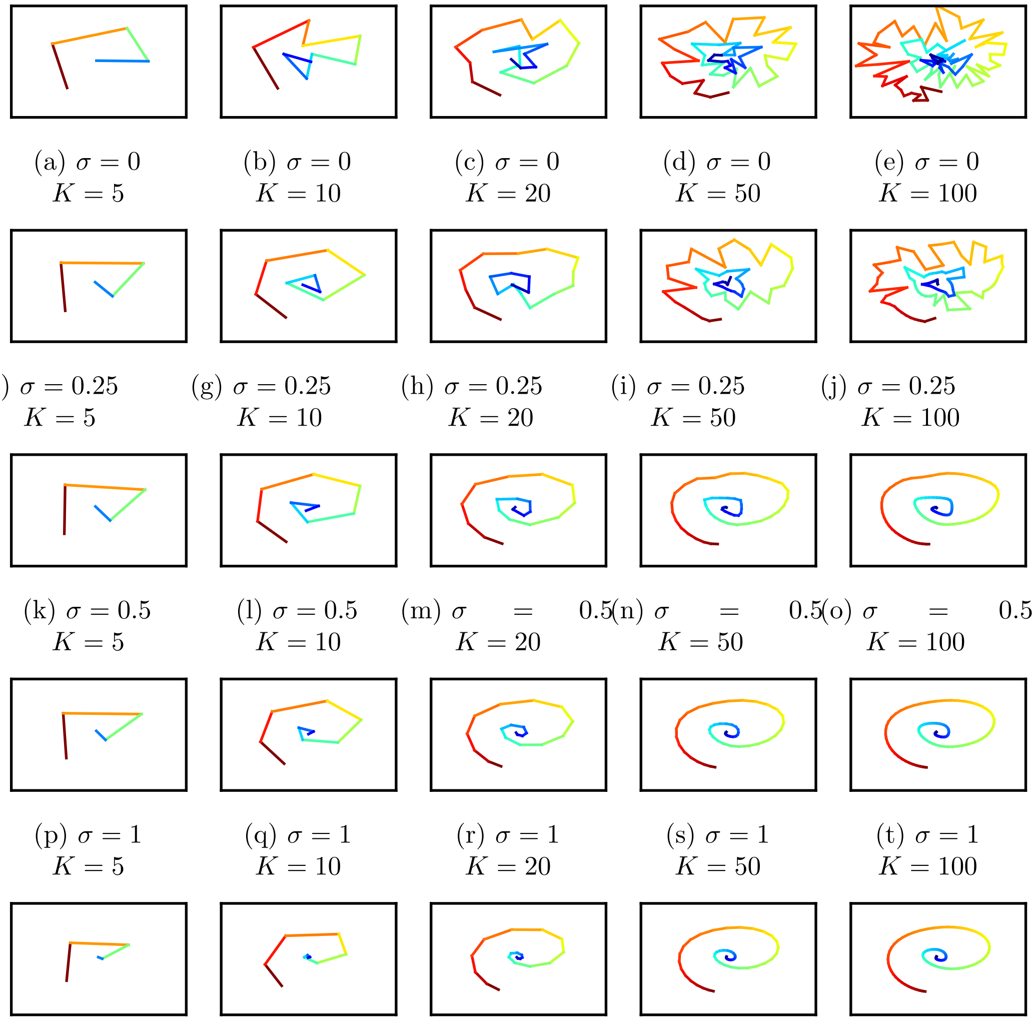
Summarized in terms of this thesis' major research questions, the results of the mouse gesture study indicate that OMMs provide an interpretable prototype **(Q6)** after only few iterations in training **(Q8)**. Choosing adequate hyperparameters is crucial for quality and smoothness of the prototype **(Q7)**: small values of the deviation parameter σ yield noisy prototype representations, while a too large value might eliminate details. The number of model states controls the resolution of the prototype time series.

7.3. Scenario: Gesture Imitation for Virtual Agents

In order to further demonstrate the prototype property of OMMs we will outline and analyze an application in which OMM prototypes are used for imitation learning of arm gestures. In this application, we understand gestures as expressive wrist movements that are executed in free space. The setup is optimized towards imitation learning during human-agent interaction between the humanoid virtual agent VINCE and a human user. Thereby, the agent imitates the gestures the user performs by means of learned OMM prototypes.

7.3.1. Data Set

The setup comprises a time-of-flight camera, a marker-free tracking software and a humanoid virtual agent called VINCE (see Figure 7.6). The time-of-flight camera (a SwissRanger$^{\text{TM}}$



(a) $\sigma = 0$ $K = 5$

(b) $\sigma = 0$ $K = 10$

(c) $\sigma = 0$ $K = 20$

(d) $\sigma = 0$ $K = 50$

(e) $\sigma = 0$ $K = 100$

(f) $\sigma = 0.25$ $K = 5$

(g) $\sigma = 0.25$ $K = 10$

(h) $\sigma = 0.25$ $K = 20$

(i) $\sigma = 0.25$ $K = 50$

(j) $\sigma = 0.25$ $K = 100$

(k) $\sigma = 0.5$ $K = 5$

(l) $\sigma = 0.5$ $K = 10$

(m) $\sigma = 0.5$ $K = 20$

(n) $\sigma = 0.5$ $K = 50$

(o) $\sigma = 0.5$ $K = 100$

(p) $\sigma = 1$ $K = 5$

(q) $\sigma = 1$ $K = 10$

(r) $\sigma = 1$ $K = 20$

(s) $\sigma = 1$ $K = 50$

(t) $\sigma = 1$ $K = 100$

(u) $\sigma = 2$ $K = 5$

(v) $\sigma = 2$ $K = 10$

(w) $\sigma = 2$ $K = 20$

(x) $\sigma = 2$ $K = 50$

(y) $\sigma = 2$ $K = 100$

Figure 7.4.: The plots illustrate the quality of the learned prototype in dependency on the chosen OMM hyperparameters. As indicated in the caption of each plot, the rows show models for different value of the deviation parameter σ and models in the columns are associated to the number of model states.

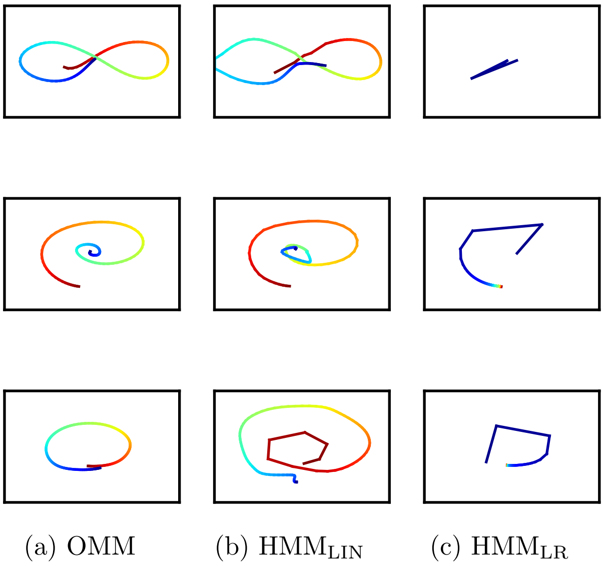


Figure 7.5.: These plots show the learned prototype representations based on OMMs, HMMs$_{\text{LR}}$, and HMM$_{\text{LIN}}$.

SR4000[2]) captures the scene in 3D at a frequency of ≈ 30 fps. The software iisu$^{\text{TM}}$ 2.0[3] then maps a human skeleton on the present user in the scene. We extract the relevant information of the skeleton, such as the user's height, spatial positions of the wrists and the center of mass. We compute normalized 3D positions of the wrists with respect to the user's body size. Within a body-correspondence-solver module, the wrists' positions are transformed (rotated and scaled) from the coordinate system of the camera to egocentric space of the virtual agent which stays face-to-face to the human demonstrator. Our focus in the current study lies on the location of the right wrist. The data was recorded as time series for each performed gesture. During data acquisition, VINCE imitates the subject's right hand movements in real time. In this way, the demonstrator receives visual feedback on how VINCE performs the shown gestures. The ambiguous position of the elbow at each time step is not captured but computed with the aid of inverse kinematic [Tolani et al., 2000]. We captured 520 examples gestures in 48 classes in the format of 3D wrist location coordinates trajectories with time stamps. Each trajectory starts from and ends at the rest position of the right hand, whereas the gestures were demonstrated at different velocities and require an average execution of 4.75 seconds. The performed gestures ranged from conventional communicative gestures (*waving*, *come here*, and *calm down*) over iconic gestures (*circle*, *spiky*, *square*, *surface*, and *triangle*) to deictic gestures (*pointing* to left, right, and upwards). These gestures have been performed as 48 different classes, each with respect to some of the following variant features: size (e.g., small and big circles), performing space (e.g., drawing a circle at the right side or in front of oneself), direction (clockwise or counter-clockwise), orientation (horizontal or vertical), repetition (repeating some sub-parts of the movement, such as drawing a circle once or twice, or swinging the hand for several times during waving). Figure 7.7 shows graphical representations of randomly selected gestures from the classes *surface front horizontal three times*, *circle big*

[2] http://www.mesa-imaging.ch
[3] http://www.softkinetic.net

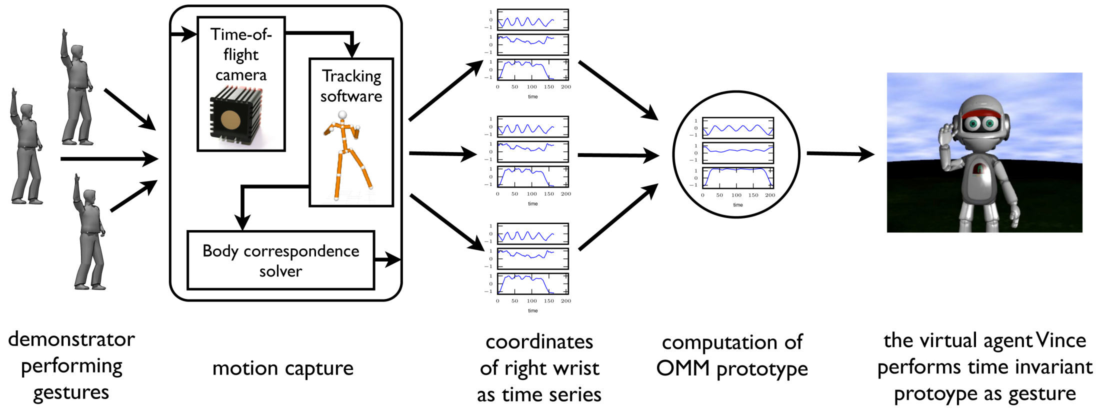

Time-of-flight camera
Tracking software
Body correspondence solver
demonstrator performing gestures
motion capture
coordinates of right wrist as time series
computation of OMM prototype
the virtual agent Vince performs time invariant protoype as gesture
time

Figure 7.6.: In our setup, demonstrated right hand gestures are captured and preprocessed within the motion capture module and the resulting 3D trajectories are stored as time series. OMM prototypes are computed from different demonstrations and performed by the virtual agent VINCE, as the result of the prototype learning process.

front counterclockwise vertical two times, *circle small right counterclockwise vertical two times*, *triangle front counterclockwise vertical one time*. The complete data set is available as supplementary material.

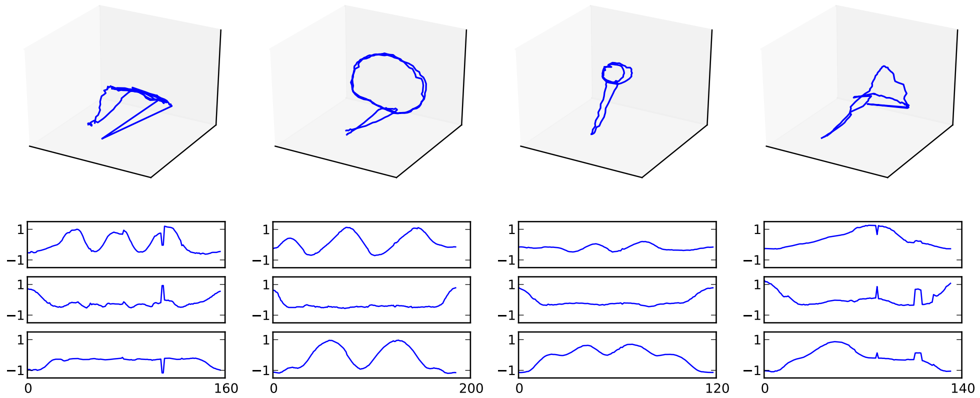
7.3.2. Experiments

In order to evaluate if OMMs are able to learn gesture prototypes we designed an experimental setup to investigate the following research questions:

1. Do the learned OMM parameters provide an interpretable prototype that can be utilized in human-agent interaction and imitation learning?
2. Can these prototypes additionally be used for recognition of unknown gestures? How do the OMMs perform in terms of generalization accuracy, even if only few training data are available?
3. What influence does the number of model states in OMMs have on the classification accuracy and the computational demands?

To address the first question, we trained an OMM for each gesture class and examined the resulting model parameters. Subsequently, we let the virtual agent VINCE execute the trained prototypes and captured these executions on video (cf. supplementary material).

In order to address the second question, we compared OMM classifiers to a standard classification technique in terms of classification accuracy and running times on artificially reduced training data sets. For comparison, we chose Nearest Neighbor Classifiers based on Dynamic Time Warping distance function [NN_{DTW}, Chiba and Sakoe, 1978, please see Section 5.1.1 for details]. We evaluated both classifiers with subsets from the training data set, whereby the amount of training data per class ranged from one to seven examples. Additionally, we conducted classification experiments with all available training data. To obtain the final error rate, we applied the resulting classifiers to the dedicated test data set.



gesture:	surface	circle	circle	triangle
repetitions:	3 times	2 times	2 times	1 time
position:	front	front	right	front
size:	-	big	small	-
direction:	-	CCW	CCW	CCW
orientation:	horizontal	vertical	vertical	vertical

Figure 7.7.: This figure shows plots of randomly selected gesture trajectory from the time-of-flight data set as 3-dimensional plots (first row) and as x-, y-, z-location coordinates varying in time (second row).

We evaluated the influence of the number of model states K in a similar way. Here, the classifiers were trained with a reduced number of model states K and number of training examples to test their generalization capabilities with the complete test data set.

For all experiments, we partitioned the data set into a training set (369 example gestures) and a test set (the remaining 151 examples). All data was normalized to zero mean and unit variance according to the training data. We identified optimal OMM hyperparameters K and σ by means of leave-one-out cross validation on the training data set. As the criterion for hyperparameter optimization, we chose the area under the receiver operating curve (AUC, cf. Section 3.6.3). We trained a model with the training data of one class (except the one left out). We then used the left out gesture as a positive and the gesture data from the remaining classes as negative examples for AUC analysis. We chose eleven equidistant values for the number of OMM states $K \in \{10, 30, \ldots, 230\}$, the set of values for the standard deviation parameter was $\sigma \in \{2 \cdot 0.75^y | y = 1, \ldots, 10\}$.

7.3.3. Results

Figure 7.8 illustrates learned gesture prototypes for gestures from class *circle big front counterclockwise vertical two times*. Each column illustrates a model that trained with different values for the number of model states K. The first row shows the data as 3-dimensional plots and the second row shows the same data as location coordinates developing over time. For the prototypes, we chose three different values for the number of states $K \in \{10, 30, 50, 100\}$. As standard deviation parameter, we chose $\sigma = 0.84375$, the value that reached the highest AUC value of ≈ 0.95 in cross validation.

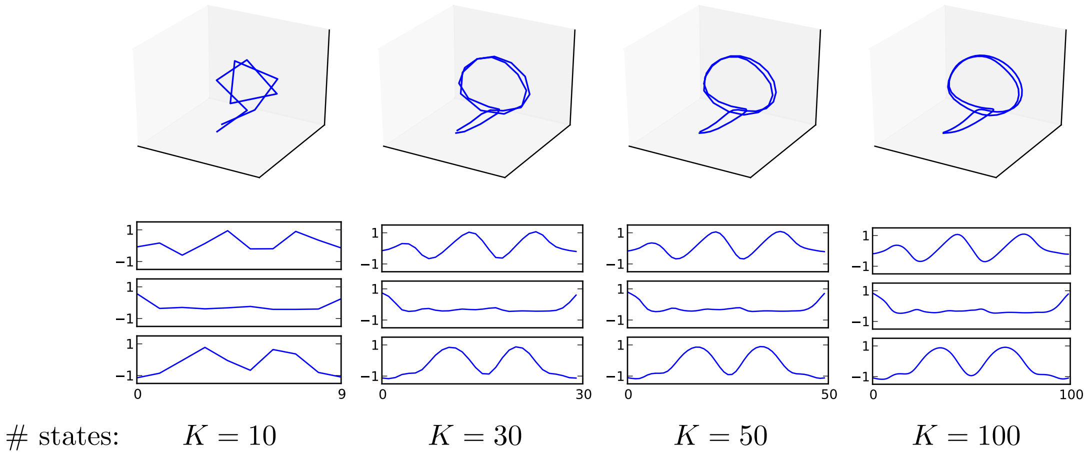
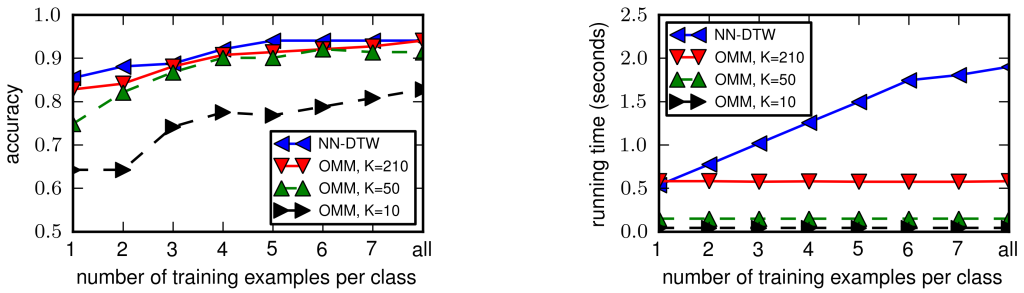


Figure 7.8.: This figure illustrates OMM prototypes trained with different values for the number of model states K, as 3-dimensional plots (first row) and as x-, y-, z-location coordinates varying in time (second row). In each column an OMM prototypes trained with all available examples from class *circle big front clockwise vertical one time* with 10, 30, 50, and 100 model states is plotted, respectively.

The plots indicate a clear correspondence between an underlying gesture class and the learned OMM parameters, and it is visually obvious that OMMs are able to extract prototype representations of the gestures. Even a prototype with $K = 10$ model states reveals a trajectory that is similar to the genuine circle gesture as displayed in the second column of Figure 7.7. For $K = 30$ the plot of the prototype fully represents a circle gesture that, in comparison to the models with $K \in \{50, 100\}$ states, only differs in length.

To underline the prototype capacity of OMMs, we executed all gesture prototypes with our virtual agent VINCE. In the supplementary material, a video is attached that contains example gestures from all 48 gesture classes, and recordings of the virtual agent performing the related gesture prototypes. Additionally, Figure 7.10 shows screenshots of these video recordings. The first row shows video screenshots from a human demonstrator performing a gesture during data acquisition; the screenshots in the second row show how the virtual agent VINCE is executing the learned OMM parameters from the matching class. In this video the demonstrator and VINCE are performing the gesture *waving head 2.5 swings*.

In general, both results — the examination of the learned prototype as well as the videos of VINCE who executes these prototype gestures — indicate that the architecture of OMMs is able to deduce essential gesture features from a set of example trajectories. However, some videos (e.g., all *come* and *surface* prototypes) suggest that using only a limited body model, such as the right hand wrist, might not be adequate to fully represent and reproduce a gesture. E.g., VINCE's performances of the *come* prototypes lack the orientation of his hand palm. Even though the plain hand wrist trajectory matches the subject's hand wrist trajectories, the incorrectly oriented palm might make it difficult for a human user to identify the intended gesture. Presumably, a more detailed body model, as, e.g., in Bergmann and Kopp [2009], would improve the prototype representation. In contrast, other gesture classes are sufficiently represented only by hand wrist trajectories, e.g., VINCE's performances of all waving related classes are easy to comprehend.



(a) Classification accuracy of test data set.

(b) Average running time for classification of one unseen gesture.

Figure 7.9.: This figure shows the accuracy (Fig. a) and average running times (Fig. b) for single trial classification of unseen gestures for NN_{DTW} and OMM classifier depending on the number of training examples per class. For OMM classifier, different values for the number of model states $K \in \{10, 50, 210\}$ are plotted.

Figure 7.9a shows the performance results of our evaluation in terms of classification accuracy on the test set depending on the number of training examples per class for different classifiers. These include OMMs with a high number of states according to maximum classification accuracy ($K = 210$) and OMMs with a reduced number of states ($K = 10, 50$). In general, all classifiers are able to recognize unseen gesture trajectories with high accuracy, although OMM classifiers with 10 model states reach substantially lower accuracy. Using all training examples, NN_{DTW} as well as OMMs classify gestures with a high accuracy of ≈ 0.94, although the performance for OMM classifiers with only $K = 10$ is noticeably lower. The plot also shows that a reduction of the number of training examples does not substantially reduce the classification accuracy. Only for OMMs with a low number of states, a degradation can be observed below three examples.

The slightly higher recognition performance of NN_{DTW} classifiers comes at the cost of substantially increased computational demands. In the scenario with all available training data, OMM classifiers provide an average speed-up factor of at least ≈ 3. For decreasing number of model states the speed-up factor increases once more. OMM classifier with $K = 50$ respond in ≈ 0.14 seconds, with $K = 10$ OMMs classify an unseen gesture in 0.04 seconds. In comparison to the average classification times of NN_{DTW}, this is an acceleration between 3 and 44 times. This allows low-latency recognition of gesture performances, which is a requirement for interaction with humans. Please note that HMM_{LR}-based classifiers would likely reveal a — similar to OMMs — flat dependency between the number of training examples and running time. However, due to an algorithmic complexity that is polynomial in the number of model states, the computational demands would be substantially higher than the one of OMMs.

In lights of the **major research questions** of this thesis, the results from these experiments further consolidate the findings from our mouse gesture experiments (Section 7.2): (i) OMMs are able to reveal an interpretable prototype for gesture data that can, e.g., be used for imitation learning **(Q6)**. (ii) The chosen hyperparameters are crucial for the prototype representation **(Q7)**, and improper selected hyperparameters might result in noisy or segmented prototype representations.

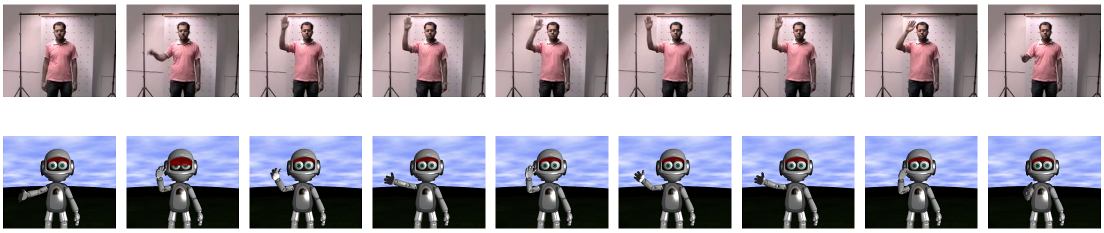


Figure 7.10.: This figure shows screen-shots from the gesture videos. The first row shows video screen-shots of a human demonstrator during data acquisition. In the second row VINCE, a virtual agent, performs the corresponding OMM prototype. The gesture in these videos is from class *waving head 2.5 swings*.

7.4. Summary

The aim of this chapter was to evaluate the capabilities of OMMs for reproduction of interaction data as, e.g., needed in many communication scenarios or for imitation learning with robots and agents. We focused on investigating of OMM's prototype property, the influence of the hyperparameter selection and the training process on the emerging prototypes.

Initially, in **Section 7.1** we addressed the term prototype and described how it is understood in this thesis. Additionally, we distinguished its definition from the widely used term model. Here, we emphasized that, in contrast to models, prototypes are of the same type as the observations it ought to be prototypical for.

Subsequently, we analyzed a mouse gesture data set in **Section 7.2**. The data set comprises 33 computer mouse gestures in three classes (*spiral*, *figure eight*, *circle*) performed by 11 human participants. Since we chose to represent computer mouse gestures as 2-dimensional location coordinates with time stamps, we were able to easily inspect the gestures as well as the OMM prototypes visually. Analyzing the learned OMM prototype representations for the computer mouse gestures revealed that the model parameters of OMMs are well suited to reproduce the gestures. Depending on the chosen model configuration, OMMs can be used as prototypes for the given gestures classes. Thereby, OMM's deviation parameter σ controls the smoothness of the prototype with higher values of σ yielding smoother prototypes. The number of model states K governs the sampling rate of the prototype, i.e., the pace or the frequency. In comparison to reproduction capabilities of similar HMMs, the results indicate that OMMs are able to reveal the relevant information quicker. However, depending on the chosen model topology, HMMs did also provide a feasible prototype.

In **Section 7.3** we applied OMMs to reproduce and recognize natural gesture trajectories. Here, we captured 48 hand-arm gestures as 3-dimensional location coordinates of the right hand wrist, with 520 collected gestures performances. In this scenario, we used a Time-Of-Flight camera in combination with a tracking software and a body-correspondence solver to capture the gesture data. The results from our classification experiment showed that OMMs are able to learn gestures from multivariate time series even if only few observations are available. Furthermore, our run time measurements indicate, that OMMs are well suited for low latency gesture recognition. Even though more complex models and methods might further increase the recognition performance, in particular in human computer interaction scenarios the response time is crucial. Here, OMMs are able to provide a suitable trade-off between accuracy and computational demands. We showed that OMMs with few model states can still reach competitive accuracy while considerably decreasing computational demands to ensure low latency capability.

According to the **major research questions** of this thesis, the results from these experiments indicate that OMMs provide a prototype representation for interaction data that is represented as time series. These prototypes can be used for imitation learning during human-agent interaction **(Q6)**. In addition, the choice of hyperparameter values influences the quality and characteristic of the learned prototype. Incorrectly selected hyperparameters might result in noisy or segmented prototypes **(Q7)**. Further results suggest that the iterative training scheme of OMMs is able to quickly reveal the relevant information for a prototype from a data set. In some cases, only one iteration was necessary to expose a prototype from an interaction time series data set **(Q8)**. In terms of running time, the results from a study in gestural human-agent interaction indicate that OMM-based classifiers are able to provide a suitable trade-off between accuracy and computational demands **(Q2)**.

8. Organization of Interaction Time Series Data with OMMs

Besides recognition (cf. Chapter 6) and reproduction (cf. Chapter 7) of observations, desirable properties for interactive system are organization capabilities of complex behavior observations according to an inherent structure. Humans, who observe meaningful interactions such as arm gestures, are easily able to incrementally and abstractly structure arising observations by common characteristics such as, e.g., orientation, shape, size, or velocity. In addition, humans reproduce these observations with their own body according to the learned structures — and a human observer is capable to rapidly generalize from these characteristics and derive an abstract meaning independently from its properties. This is, e.g., the case in humans that communicate by means of sign languages: the interactants are able to structure and organize sign language signs, and they make sense of their observations in an incremental way, even though sign language signs cover substantial inter-personal variations.

A computer system that is to participate in such interactions would greatly benefit from being able to organize its observations according to the inherent data structure. E.g., if the human user performs a behavior pattern from one category in very different ways, the system should be able to structure the performances accordingly. This gives rise to the following research questions:

- **(Q9) Organization:** How can OMMs be used to organize observations that are represented as time series in an unsupervised manner, without intervention of an external human observer?
- **(Q10) Hierarchical Organization:** How can we enable OMMs-based methods to discover inherent but hidden hierarchical structures in observed interaction behaviors, again unsupervised?

According to these questions, ideally, a system should be able to identify relevant similarities, dissimilarities, and, thus, data clusters of hierarchical or non-hierarchical structures without user interventions.

In this chapter, we will investigate if OMMs are able to organize interaction time series data in an unsupervised manner. We will introduce and evaluate two approaches to unsupervised time series processing based on OMMs. First, we will present and evaluate $\mathcal{K}$-OMMs, a clustering solution for time series that is similar to classical $\mathcal{K}$-means clustering. Second, we will assess $\mathcal{K}$-OMM-trees, a tree-based approach to unsupervised hierarchical organization of time series.

This chapter is organized as follows: after introducing the $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-trees algorithms in Section 8.1, we will, first, evaluate 20 benchmark data sets from the UCR time series repository [Keogh et al., 2006] in Section 8.2. In addition to address the clustering performance of $\mathcal{K}$-OMMs algorithm, we will discuss different initialization strategies for

$\mathcal{K}$-OMMs clustering. Subsequently, we will compare the $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-trees clustering approaches by means of interaction time series data sets from the UCI machine learning repository. Last, in Section 8.4, we will recall the human-agent interaction scenario that was introduced in Section 7.3. Here, we will show that the unsupervised organization capabilities of $\mathcal{K}$-OMM-trees allow an improved straightforward generalization in context of gesture recognition.

8.1. Methods: Clustering of Time Series Data with OMMs

In order to realize organization of time series, we present two clustering algorithms for time series data. First, $\mathcal{K}$-OMMs reveal flat partitions in a data set, while $\mathcal{K}$-OMM-trees expose hidden hierarchical structures.

8.1.1. Partitioning Clustering: $\mathcal{K}$-OMMs

To discover subsets in a set of observations in an unsupervised manner, we adapt a $\mathcal{K}$-means procedure, in which the cluster centroids are replaced by OMM prototypes Ω and the negative production likelihood $p(O|\Omega)$ is applied as a distance function. $\mathcal{K}$-means clustering is a commonly used method to partition a set of observations into K groups that are associated by a joint quality. The general idea of $\mathcal{K}$-means is to minimize an objective function that accumulates the distances between each observation O^i and its nearest cluster centroid C^j to which it is assigned to. Please see Section 3.4.1 for a detailed introduction of $\mathcal{K}$-means clustering.

If the observations are multivariate time series, specific time series distance functions and centroid representations are required. To apply OMMs to clustering analysis, a feasible distance function between a time series O^i and a model Ω^j is given by the negative production log-likelihood

$$d(O^i, \Omega^j) = -\ln p(O^i|\Omega^j). \tag{8.1}$$

Please note that since such a distance function is unsymetric, it does not define a distance measure. If the Ω^j represent the cluster centroids, the objective function of $\mathcal{K}$-OMMs accumulates the likelihoods that the time series O^i have been generated by their associated centroid models Ω^j. As introduced in Chapter 7, the learned models that are used as centroids can be interpreted as prototypes for the set of associated observations. The update function in such an approach is a re-training of the models according to the assigned time series. As introduced in Chapter 7, the learned models that are used as centroids can also be interpreted as prototypes for the set of associated observations. Note that in case of $\mathcal{K}$-means the number of prototypes $\mathcal{K}$ has to be determined by an external process such as the elbow method.

Pairwise Sequence Distance

As described above, a common distance function between a time series and an OMM is the negative production log-likelihood. Since an OMM ultimately is a time series itself, i.e., an array of reference vectors, this definition can be easily extended to a pairwise non-symmetric distance function between two time series O^i and O^j. If we consider one time series as a generating model and the other one as the time series that has been generated, a feasible distance definition between these two time series would be $d(O^i, O^j) = p(O^i|O^j)$. Note that in contrast to similar distances based on HMMs, no additional training or parameter estimation is necessary. Following the suggestions that Rabiner [1989] made for HMMs, an appropriate symmetric distance measure definition is given by

$$d_{OMM} = \frac{1}{2} \cdot \left(p(O^i|O^j) + p(O^j|O^i)\right). \tag{8.2}$$

Cluster Initialization

Due to local extrema, the quality of the $\mathcal{K}$-means solution heavily depends on initialization. A naïve and yet common approach is to randomly choose cluster assignments as initialization. For clustering of time series with HMMs, Smyth [1997] suggests an initialization based on pairwise distances between all examples of a data set. These distances can be exploited to cluster a data set hierarchically and encourage compact cluster initializations beforehand.

In this study we use, alongside randomized initial assignments of the examples to clusters, a similar approach to identify a compact initialization. We compute the symmetric distances as defined in Equation 8.2 for the time series, and we apply hierarchical clustering with the "average" merge heuristic [cf. Anderberg, 1973] to find $\mathcal{K}$ initial clusters.

8.1.2. Hierarchical Clustering: $\mathcal{K}$-OMM-trees

To enable hierarchical identification of time series prototypes, we use $\mathcal{K}$-OMM-trees — an approach with structurally similar properties as $\mathcal{K}$-trees [De Vries et al., 2009, Geva, 2000]. Generally, $\mathcal{K}$-OMM-trees are height balanced clustering trees that combine classical $\mathcal{K}$-means and B+-trees algorithms with OMMs to clustering trees for time series data. A $\mathcal{K}$-OMM-tree of order m is defined as [Geva, 2000]:

1. All leaves are on the same level.
2. All internal nodes, including the root, have at most m children, and at least one child.
3. Clusters centroids are represented as OMMs and are used as search keys.
4. The number of keys in each node is equal to the number of its nonempty children, and these keys partition the keys in the children to form a search tree.
5. Leaf nodes contain observed time series.

The time complexity of building $\mathcal{K}$-OMM-tree is — similar to standard $\mathcal{K}$-trees — $O(n \log n)$. The nodes of the tree are represented by OMM prototypes and the leafs hold the time series examples. The process of building a $\mathcal{K}$-OMM-tree is similar to the building of a $\mathcal{K}$-tree. However, in contrast to Euclidean distances, we use the negative log-likelihood (cf. Eq. 8.1) as a distance function and re-train the OMMs if a new time series is added to a branch. Since OMMs are represented by ordered arrays of reference vectors and are, therefore, time series themselves, it is possible to build "OMMs of OMMs", i.e., to train a model for a set of models. This unique property enables a fast training of $\mathcal{K}$-OMM-trees: for nodes that contain sub-trees, it is possible to build an OMM representation out of the OMM representations from the level below.

Building a $\mathcal{K}$-OMM-tree is a dynamic process that inserts time series on-line, as they are observed. Consider an already existing $\mathcal{K}$-OMM-tree of order m to which an additional time series O is presented for insertion. The insertion procedure is: first, the tree is searched to identify the node that contains the time series' nearest neighbor. If this node stores less than m — the tree's order — observations, O is inserted as a leaf. Since this insertion changes the cluster structure, the parent nodes up to the root node must update their search keys in terms or re-computing their associated OMMs. When a leaf stores $m + 1$ time series, it cannot contain any more observations. It then is split by applying the $\mathcal{K}$-OMMs algorithm with two clusters to the $m + 1$ elements of that node. The resulting OMMs become the search keys for two new child nodes. Now consider that the parent of these new nodes contains $m + 1$ search keys after the splitting process, too. Then it also has to be split, replacing itself in its parent by two new child nodes. This process is repeated in a similar manner until a node with less than $m + 1$ search keys is reached or a new root node replaces the current root node. The procedure of building a $\mathcal{K}$-OMM-tree is initialized with an empty root node and just one leaf.

To use $\mathcal{K}$-OMM-trees for classification, i.e., to assign an unseen gesture to one of J classes, J class-specific $\mathcal{K}$-OMM-trees are first estimated from the data. Similar to classification with OMMs as described in Section 6.1, and assuming equal prior probabilities, an unknown gesture O is assigned to the class associated with the model that yields the highest production likelihood $p(O|\Omega)$. In case of $\mathcal{K}$-OMM-trees, all models Ω that are related to the tree and represent the tree's nodes, are used for classification.

8.2. Experiment: Partitioning of Time Series Data

Since the hierarchical organization scheme of $\mathcal{K}$-OMM-trees relies on the non-hierarchical $\mathcal{K}$-OMMs approach, we will initially investigate if the latter is able to adequately discover clusters in times series data. We designed an experimental setup to evaluate:

- How do $\mathcal{K}$-OMMs perform compared to standard methods for clustering of time series data?
- What is the impact of the initialization strategies in terms of clustering performance?
- Do the learned $\mathcal{K}$-OMMs centroids provide an interpretable prototype?

8.2.1. Experiments

To answer these questions, we conducted a comprehensive study on 20 time series clustering benchmark data sets from the UCR time series repository [Keogh et al., 2006]. To cover a wide range of time series types and research scenarios, the benchmark data set collection combines examples from various research fields (cf. Section 5.6). In this scenario, we tested the following approaches:

- **$\mathcal{K}$-means:** the standard $\mathcal{K}$-means algorithm with Euclidean distance measure,
- **$\mathcal{K}$-OMMs:** the $\mathcal{K}$-means clustering where OMMs are used as cluster centroids (cf. Section 8.1.1).

Note that $\mathcal{K}$-means with Euclidean distances requires data examples of equal lengths and dimensionality. Usually, time series do not meet this criterion, but fortunately, the time series provided by the UCR repository share the same lengths. This allows us to compute the Euclidean distance between two examples and, thus, to compare $\mathcal{K}$-OMMs and standard $\mathcal{K}$-means.

Additionally, we investigated to which extent an initialization schemes changes the performance of our clustering approaches. Therefore, we tested $\mathcal{K}$-means and $\mathcal{K}$-OMMs clustering with

- **Random initialization:** the time series are randomly assigned to a cluster,
- **Compact initialization:** time series are merged to initial clusters by similarity according to a pairwise distance as described in Section 8.1.1.

Since ground truth is available, i.e., all data sets are fully labeled, we applied purity as a validation measure to the learned cluster assignments (Section 3.6.4). Note that we did not estimate the number of clusters in this study; instead, we assumed to know the right amount beforehand.

We applied a uniform procedure to all data sets. First, we estimated an appropriate hyperparameter for each method by means of the training data as described in Section 5.2. For OMMs we tested 10×11 hyperparameter pairs where $K \in \{T/10, 2 \cdot T/10, \ldots, T\}$ with T being the time series lengths of the particular data set, and $\sigma \in \{1.5 \cdot 0.75^y | y = 0, \ldots, 9\} \cup \{0\}$. Subsequently, we took the optimal hyperparameters found in this process to train a $\mathcal{K}$-OMMs model with the complete training data set and applied the resulting clustering models to the test data set.

8.2.2. Results and Discussion

Table 8.1 displays the optimal hyperparameters found for all data sets. K denotes the number of model states, and σ the standard deviation parameter that yields the highest clustering purity for the benchmark's training sets. For some data sets (*Two_Patterns*, *Beef*, *wafer*, *Lighting7*, *Gun_Point*, *CBF*, *OSULeaf*, *Trace*) the best models found require considerably fewer model states K than the time series' length T to represent the data. In contrast, data sets like *Coffee*, *FaceFour*, *SwedishLeaf*, and *EGC200* used exact $K = T$ states to model the data, which averages to one state per sample.

Table 8.2 shows the clustering purity values for the algorithms and initialization schemes achieved by means of the benchmark's testing data sets. For the majority of data sets, $\mathcal{K}$-OMMs reach higher purity values than standard $\mathcal{K}$-means. With random initialization $\mathcal{K}$-OMMs yield higher purity levels for 13 data sets, while $\mathcal{K}$-means is superior on seven data sets. The results for the compact initialization approach uncover a similar picture: while $\mathcal{K}$-OMMs yield superior purity values for 12 data sets, $\mathcal{K}$-means reach a higher purity for eight data sets. In particular, $\mathcal{K}$-OMMs are able to represent the *Two_Patterns*, *wafer*, *50words*, and *CBF* data sets better as compared to standard $\mathcal{K}$-means. Independently of the chosen initialization, the performances differ up to $\approx 30\%$. $\mathcal{K}$-means only reaches higher purity values for the *yoga* and the *OliveOil* data sets where $\mathcal{K}$-OMMs are outperformed by more than $\approx 10\%$.

If we focus on the evaluated initialization strategies, we only find notable differences for very few data sets. While the compact initialization in comparison to random initialization enhances the results of standard $\mathcal{K}$-means for ten data sets, the performances are reduced or remain constant for the other ten data sets. For $\mathcal{K}$-OMMs the performance only increases for eight data sets, the purity decreases for seven data sets, and remains equal for five

data set	K	σ
Two_Patterns	38	0.844
Coffee	28	0.633
yoga	341	0
Beef	94	0.356
wafer	15	0
OliveOil	285	1.5
Lighting7	64	1.125
FaceFour	350	1.5
SwedishLeaf	128	0.844
FaceAll	105	0.475
50words	162	0.844
Gun_Point	15	0.113
ECG200	96	1.5
CBF	13	1.125
OSULeaf	85	0.633
Adiac	70	0.475
synthetic_control	48	1.125
Lighting2	318	0.267
fish	417	0.633
Trace	28	0.475

Table 8.1.: This table displays the optimal hyperparameter pairs found during training set evaluation of the benchmark data sets from the UCR repository. K denotes the number of model states, and σ is the standard deviation parameter that is associated with the emission densities of OMMs.

data sets. In particular, the performances of $\mathcal{K}$-OMMs strongly improve for *Two_Pattern*, *FaceAll*, *synthetic_control*, and *fish* data sets with enhanced purity between ≈ 10% and ≈ 20%. For standard $\mathcal{K}$-means the only notable performance change is a decreased purity value for the *Trace* data set by ≈ 12%. For the remaining data sets, the performance differences in this scenario are only marginal.

Figure 8.1 illustrates the cluster centroids as prototypes alongside randomly chosen examples from the *synthetic_control* data set. Each row corresponds to one class of the data set. The left column displays the time series and the right column shows the associated model. The *synthetic_control* data set contains simulation data in six classes from a domain of control chart patterns. These classes correspond to typical developments in control charts, namely, *normal*, *cyclic*, *decreasing trend*, *increasing trend*, *upward shift*, and *downward shift*.

In the examined scenario with 20 benchmark data sets, we found that $\mathcal{K}$-OMMs are generally able to discover appropriate hidden structures. For the majority of data sets, $\mathcal{K}$-OMMs could reach higher purity values than standard $\mathcal{K}$-means — up to a difference of more than 30%. Unfortunately, there is little knowledge about the particular characteristics of each benchmark data set (e.g., if it is non-stationary, noisy, etc.) and we cannot further interpret these results, e.g., to point out particular time series properties for which $\mathcal{K}$-OMMs are especially well suited.

data set	random initialization		compact initialization	
	$\mathcal{K}$-means	$\mathcal{K}$-OMMs	$\mathcal{K}$-means	$\mathcal{K}$-OMMs
Two_Patterns	0.48	**0.61**	0.47	**0.81**
Coffee	**0.76**	0.72	**0.77**	0.72
yoga	**0.81**	0.69	**0.86**	0.68
Beef	0.57	**0.62**	**0.64**	0.57
wafer	0.80	**0.94**	0.8	**0.94**
OliveOil	**0.85**	0.70	**0.82**	0.70
Lighting7	**0.72**	0.63	**0.67**	0.63
FaceFour	0.77	**0.80**	0.68	**0.73**
SwedishLeaf	**0.63**	0.59	**0.67**	0.47
FaceAll	0.43	**0.74**	0.49	**0.85**
50words	0.45	**0.74**	0.51	**0.75**
Gun_Point	0.70	**0.73**	0.70	**0.73**
ECG200	**0.84**	0.81	0.79	**0.82**
CBF	0.73	**0.94**	0.76	**0.95**
OSULeaf	0.51	**0.57**	**0.59**	0.54
Adiac	**0.60**	0.49	**0.61**	0.55
synthetic_control	0.85	**0.87**	0.85	**0.99**
Lighting2	0.75	**0.80**	0.75	**0.80**
fish	0.60	**0.73**	0.58	**0.60**
Trace	0.70	**0.73**	0.70	**0.85**

Table 8.2.: This table displays the purity values of the benchmark test data sets for $\mathcal{K}$-OMMs, $\mathcal{K}$-means and the initialization variants. The bold font indicates the highest purity reached for the particular initialization approach.

The findings of the initialization experiment reveal an ambiguous picture. On one hand, the compact initialization strategy heavily enhances the clustering accuracy for some data sets. On the other hand, the majority of data sets could not benefit from such an approach. Surprisingly, even standard $\mathcal{K}$-means with Euclidean distances could not take advantages out of the compact initialization scheme. This indicates that the chosen initialization strategy might not be well suited for this method or this type of data in general.

The plots of the model parameters in Figure 8.1 indicate that centroid OMMs are good prototypes for the learned data partitions. The array of model parameters as plotted in the right column of Figure 8.1 allow further insights in the data, in particular, if no or only little prior knowledge about the data set is available. A clear correspondence between the time series and the associated learned models is visible. Moreover, the models in the right column nicely fit the underlying structures of the classes as described in Section 8.2.2. The class *cyclic* displayed in the second row is particularly interesting as the number of cycles between the example sequence (4 maxima) and the learned prototype (5 maxima) differs. This indicates that other observations from this class might have a different cycle frequency as compared to the randomly selected one. The model represents this characteristic in a prototype that consolidates all observations according to the OMM data representation.

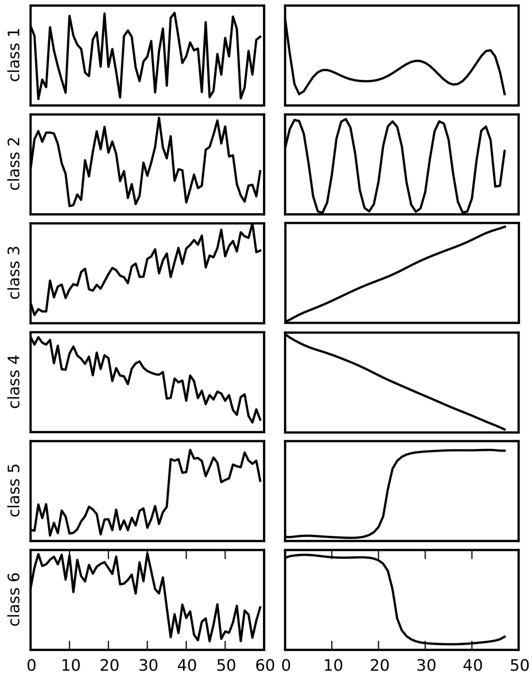


Figure 8.1.: Examples from the *synthetic_control* data set and corresponding OMMsṪhe left column shows one randomly chosen time series from each class. The right column shows the uncovered OMM prototype that matches the particular example.

In general, the $\mathcal{K}$-OMMs approach to clustering of time series might be particularly interesting in situations where only little or no knowledge about the data is available. Then, $\mathcal{K}$-OMMs can discover hidden structures and present informative cluster centroids of these structures by means of time series prototypes.

In terms of the major research questions of this thesis, theses results show that $\mathcal{K}$-OMMs are able to uncover hidden clusters in time series data sets **(Q9)** with high accuracy. Further findings indicate that the learned OMM cluster centroids are appropriate prototypes for particular data clusters **(Q6)**, which allows additional insights into an analyzed time series data set.

8.3. Experiments: Hierarchical Organization with $\mathcal{K}$-OMM-trees

To investigate if the proposed methods for clustering of time series are also able to analyze interaction data, we evaluated $\mathcal{K}$-OMM-trees and $\mathcal{K}$-OMMs with common benchmark data sets from the UCI machine learning repository.

data set	measure	$\mathcal{K}$-OMMs	$\mathcal{K}$-OMM-trees	$\mathcal{K}$-HMMs
Australian Sign Language Signs	running time	220.6s	24.5s	1462.8s
	V-measure	0.869	0.899	0.888
	homogeneity	0.779	0.83	0.811
	completeness	0.988	0.986	0.987
Character Trajectories	running time	668.4s	79.5s	2193.5s
	V-measure	0.976	0.973	0.777
	homogeneity	0.957	0.951	0.649
	completeness	0.997	0.996	0.998

Table 8.3.: Results for clustering evaluation for time series data sets from the UCI machine learning repository.

8.3.1. Experiments

In order to evaluate the $\mathcal{K}$-OMM-trees algorithm, we designed an experimental setup for investigation of the following questions:

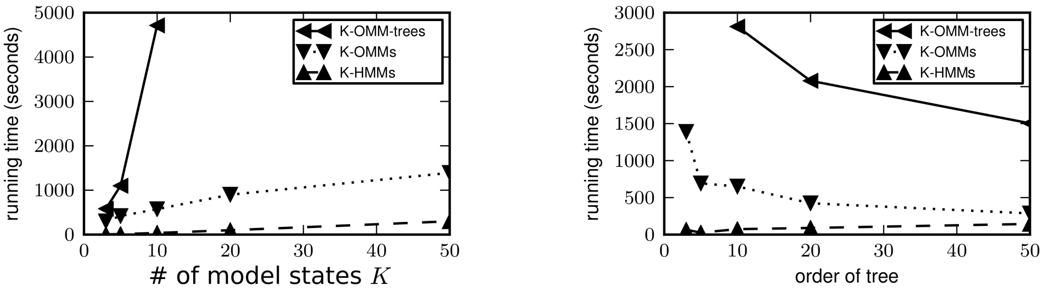
1. Can $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-tree algorithms extract clusters from interaction time series sets?
2. Are $\mathcal{K}$-OMM-trees able to extract meaningful prototypes for interaction data?

We compared $\mathcal{K}$-OMM-trees and $\mathcal{K}$-OMMs clustering algorithms on benchmark interaction data sets from the UCI machine learning repository [Frank and Asuncion, 2010]. As a base-line approach, we use a $\mathcal{K}$-HMMs clustering algorithm as described in [Perrone and Connell, 2000]. As benchmark data sets, we chose the Australian Sign Language Signs as well as the Character Trajectories data sets. For both data sets, ground truth is available and all examples are labeled; Table 5.1 shows basic properties of these data sets.

According to the first question, we evaluated the performance of the presented methods by applying common clustering evaluation measures to the obtained data clusters. We applied uniform procedures to all data sets and methods. We tested $\mathcal{K}$-OMM-trees, $\mathcal{K}$-OMMs, and $\mathcal{K}$-HMMs with different hyperparameter values. The set of values for the standard deviation parameter was $\sigma = 1$, the number of model states was chosen from $K \in \{3, 5, 10, 20, 50\}$, and the order of the $\mathcal{K}$-OMM-tree was chosen from $m \in \{3, 5, 10, 20, 50\}$. As evaluation measures we used completeness, homogeneity, V-measure, and the computational demands in terms of running time.

In order to address the second question, we used $\mathcal{K}$-OMM-trees to analyze a subset of the Character Trajectories data set. We chose 25 examples in five classes to train a $\mathcal{K}$-OMM-trees and examined the resulting model parameters. In this scenario, we chose the number of OMM states $K = 50$, the standard deviation parameter σ was set to 1, and the order of the $\mathcal{K}$-OMM-trees was 3.

For all experiments, we initialized the time series clusters with randomized cluster assignments.



(a) K

(b) *order*

Figure 8.2.: Running time of $\mathcal{K}$-OMMs, $\mathcal{K}$-HMMs and $\mathcal{K}$-OMM-trees in dependency of the number of model states and the order of the $\mathcal{K}$-OMM-tree for the Character Trajectory data set.

8.3.2. Results and Discussion

Table 8.3 shows the results of our benchmark study in terms of clustering evaluation measures and running time. In general, all methods show sufficient clustering performances related to ground truth on both evaluated data sets, i.e., almost all methods are able to discover similar cluster assignments as compared to the known class labels. $\mathcal{K}$-OMM-trees reach the highest V-measure value of ≈ 0.899, closely followed by $\mathcal{K}$-HMMs and $\mathcal{K}$-OMMs with V-measure values of ≈ 0.888 and 0.869, respectively. The homogeneity measures are similarly ranked: $\mathcal{K}$-OMM-trees, $\mathcal{K}$-OMMs, and $\mathcal{K}$-HMMs reach average homogeneity values ≈ 0.83, ≈ 0.811, and ≈ 0.799. For completeness, we find a reversed ranking. Here, K-OMMs reach the highest average completeness value of ≈ 0.988, followed by $\mathcal{K}$-HMMs and $\mathcal{K}$-OMM-trees, each with a 0.001 decreased completeness value. The results for the character trajectory data sets are different. $\mathcal{K}$-OMM-trees and $\mathcal{K}$-OMMs archive high average V-measure values of ≈ 0.973 and ≈ 0.976; in contrast, $\mathcal{K}$-HMMs only reach a V-measure value of ≈ 0.777. Similarly, $\mathcal{K}$-HMMs reach a substantially lower homogeneity value of ≈ 0.649 ($\mathcal{K}$-OMM-trees: ≈ 0.951, $\mathcal{K}$-OMMs: ≈ 0.957) in analysis the Character Trajectory data set. In addition, all methods archive very high completeness values of more than 0.99.

These results indicate that all evaluated methods are able to extract clusters in motion data sets with high accuracy. In terms of ground truth, $\mathcal{K}$-OMMs, $\mathcal{K}$-OMM-trees, and $\mathcal{K}$-HMMs are able (i) to assign examples to clusters that are members of the same class (as indicated by high homogeneity values), and (ii) to assign examples of a given class to a single cluster (as indicated by high completeness values). Only the application of $\mathcal{K}$-HMMs to the character trajectory data set leads to substantially lower cluster completeness as compared to $\mathcal{K}$-OMM-trees and $\mathcal{K}$-OMMs, however, in terms of homogeneity also $\mathcal{K}$-HMMs are able to discover appropriate clusters in the given data set.

In terms of computational running time, the methods differ substantially. In average, $\mathcal{K}$-OMM-trees require $\approx$ 25s to process the Australian Sign Language Signs data set. The $\mathcal{K}$-OMMs approach needs $\approx$ 9 times more computational power ($\approx$ 222s), whereas $\mathcal{K}$-HMMs require more than 60 times longer running time ($\approx$ 1462s) than $\mathcal{K}$-OMM-trees. A similar picture unfolds for the character trajectory data set. Here, $\mathcal{K}$-OMMs require $\approx$ 8,

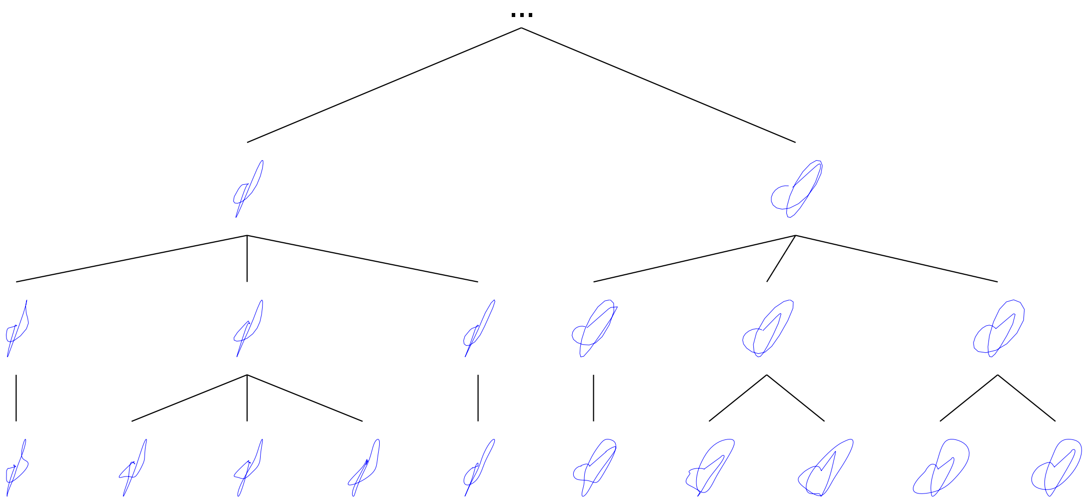


Figure 8.3.: A $\mathcal{K}$-OMM-tree learned from the Character Trajectory data set. The trajectories in the leaf are examples from the data set, the nodes contain learned OMM prototypes.

and $\mathcal{K}$-HMMs ≈ 27 times more running time as compared to $\mathcal{K}$-OMM-trees.

In order to further analyze the differences in running time, Figure 8.2 displays the computational demands of the evaluated approaches depending on the number of model states (Figure 8.2(a)), and the order of the $\mathcal{K}$-OMM-tree (Figure 8.2(b)). Figure 8.2(a) indicates that the computational demands of all three methods increase for larger amount of model states, whereby in particular the running time of $\mathcal{K}$-HMMs heavily depends on this parameter. In addition, the plot shows that the running times curve of $\mathcal{K}$-HMMs is steeper than the corresponding curves for $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-trees. We disrupted the analysis of $\mathcal{K}$-HMMs for more than 10 model states due to limited computational resources.

In terms of $\mathcal{K}$-OMM-tree's order m and the accompanying number of clusters a similar picture unfolds. For a small order, $\mathcal{K}$-HMMs require considerably more computational running time than $\mathcal{K}$-OMM-trees and $\mathcal{K}$-OMMs

Figure 8.3 illustrates a $\mathcal{K}$-OMM-tree that was learned from the Character Trajectory data set. The leaf trajectories are examples from the data set, the nodes contain OMM prototypes. The tree clearly uncovers inherent data structures: e.g., the observations of the left branch are narrower than their counterparts from the right branch. And the node OMMs compellingly reflect the characteristics of their subordinate observations.

The findings of these experiments indicate that OMMs are able to unsupervised reveal both, non-hierarchical partitions **(Q9)** and hierarchical structures **(Q10)** in interaction data sets. The results show that both presented approaches — $\mathcal{K}$-OMMs as well as $\mathcal{K}$-OMM-trees — uncover these structures with high accuracy. In dependency of the chosen hyperparameters **(Q7)**, the hierarchical $\mathcal{K}$-OMM-trees allow to process larger data sets due to dramatically reduced running times, which could be interpreted as a further indicator for OMMs' low latency capabilities **(Q2)**.

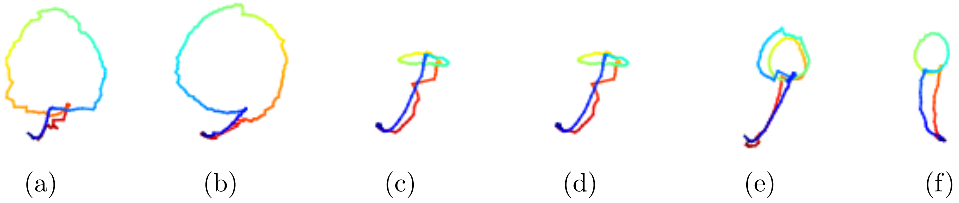


Figure 8.4.: These plots display randomly selected *circle* gestures from the data set that we used in the study. The 3-dimensional trajectories are mapped on the x/y-axis; the color indicates the time during execution (from “blue” to “red”).

8.4. Scenario: Gesture Recognition with $\mathcal{K}$-OMM-trees

To demonstrate the capabilities of our algorithm, we reuse the application setup that was introduced in Section 7.3 The scene comprises a virtual agent that learns gestures classes by means of a human demonstrator. This setting includes a high variety in gesture performances within each class.

In this study, we investigate the following questions:

1. Does the unsupervised learned structure of $\mathcal{K}$-OMM-trees support recognition of unseen gestures?
2. What influence do $\mathcal{K}$-OMM-tree’s hyperparameters have on the performance?
3. Do the learned OMM prototypes of $\mathcal{K}$-OMM-trees provide a model for gesture reproduction?

8.4.1. Data Set

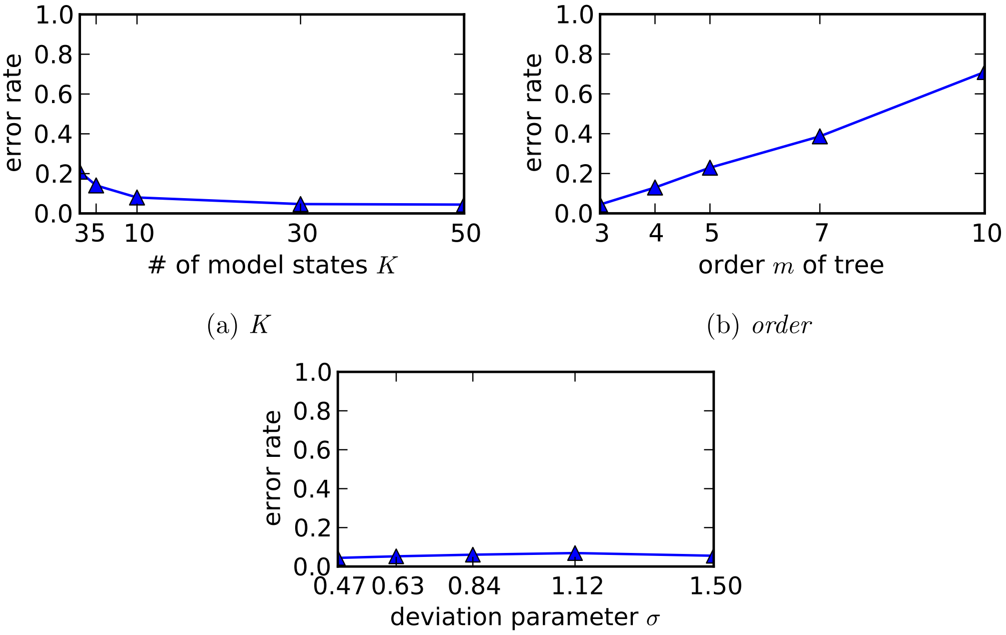
This setup employs the same gesture trajectories as the before-used Concrete Gestures data set (cf. Section 7.3). However, the trajectories are here combined differently to nine abstract gesture classes. The classes comprise gestures from higher-level semantic categories, e.g., all circle gestures are combined to one class *circle* despite of their size, performance space, direction, orientation, or repetitions. This leads to classes that enclose very different raw data trajectories, and, thus, are more challenging in terms of representation by a data model. Please see Figure 8.4 for illustrations of five randomly selected *circle* gestures recorded in this data set. The nine abstract classes include conventional communicative gesture classes (*waving*, *come here* and *calm down*), iconic gesture classes (*circle*, *spiky*, *square*, *surface* and *triangle*), as well as deictic gesture classes (*pointing*). We divided the data set into training and test sets. The training set contains approximately two-thirds of all examples (362 gestures) and the remaining 147 gesture performances are used for testing.

8.4.2. Experiments

We used OMMs, HMMs, and $\mathcal{K}$-OMM-trees for classification of abstract gesture classes. Thereby, we applied an identical procedure to all classifiers as described in Section 5.2. We chose equal values for the hyperparameters in the 5-fold cross validation scheme: the

Method	K	σ	order	cv error rate	test set error rate
$\mathcal{K}$-OMM-trees	50	0.47	3	0.05	0.09
HMMs	51	0.47	-	0.24	0.19
OMMs	50	0.47	-	0.27	0.26

Table 8.4.: This table denotes the results from the classification experiments of the arm-gesture data set for all evaluated methods. In detail, the hyperparameters (number of model states K, the deviation parameter σ, and the order of the $\mathcal{K}$-tree m) as well as the cross validation and associated test set error rates are shown.



(a) K

(b) *order*

(c) σ

Figure 8.5.: This figure shows the cross validation error rates of for varying number of model states K (sub-figure (a)), varying order of the $\mathcal{K}$-OMM-tree (sub-figure (b)), as well as varying deviation parameter σ (sub-figure (c)).

number of model states was $K \in \{3, 5, 10, 20, 50\}$; the set of values for the standard deviation parameter was $\sigma \in \{0.5, 1.0, 1.5, 2.0\}$. In case of $\mathcal{K}$-OMM-trees, we additionally chose five values for the tree's order $m \in \{3, 4, 5, 7, 10\}$.

In addition, we examined the resulting OMM parameters that represent the tree's nodes.

8.4.3. Results and Discussion

Table 8.4 displays the results from the classification experiments. In terms of cross validation and test set error rates, $\mathcal{K}$-OMM-trees classifiers clearly outperform OMM- and HMM-based classifiers by up to $\approx 22\%$. While $\mathcal{K}$-OMM-trees achieve a test set error value

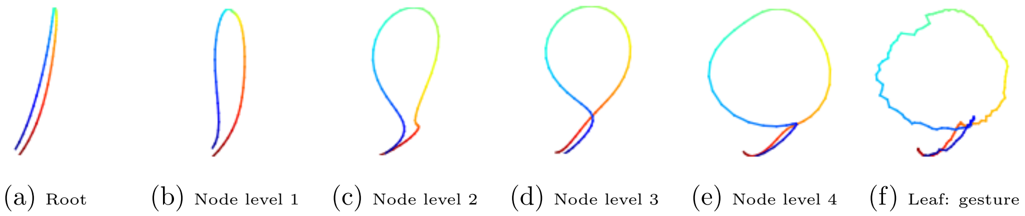


Figure 8.6.: These plots illustrate the OMM prototypes that correspond to one branch of the learned $\mathcal{K}$-OMM-tree for the class *circle*. The plots are related to child-nodes from left to right, i.e., the left-most plot is on the tree's root level, the second plot on level 1 and so forth. The plot on the very right side shows a gesture performance in a leaf. Please note that we used all three location coordinates in the training process; these figures only plot the x/y-coordinates. The trajectories are mapped on the x/y-axis; the color indicates the development in time (from "blue" to "red").

of ≈ 0.09, OMM yield ≈ 0.26 and HMMs ≈ 0.19. Similarly, the cross validation error rate of $\mathcal{K}$-OMM-trees is with a value of ≈ 0.05 substantially lower than the cross validation errors of OMMs (≈ 0.27) and HMM classifiers (≈ 0.24). These results indicate that the hierarchical structure of $\mathcal{K}$-OMM-trees induce a superior data representation, at least if, as given with the evaluated gesture data, the data set involves a wide intra-class variability.

Figure 8.5 illustrates the dependency between $\mathcal{K}$-OMM-tree's hyperparameters and the reached cross validation accuracy. Each plot displays the influence of one varying hyperparameter while the remaining parameters are fixed. Here, only a varying order of the $\mathcal{K}$-tree m leads to substantially increased cross validation error rates, e.g., the $\mathcal{K}$-OMM-tree classifier yields an accuracy of ≈ 0.71 with an order $m = 10$. In contrast, changes of the standard deviation parameter σ are almost without consequences: here, $\sigma = 1.125$ yields the highest error rate of ≈ 0.07 — only ≈ 0.02 higher than the best error rate that is archived by a classifier trained with $\sigma = 0.47$. Last, variations in the number of model states K induce small changes in cross validation performance; in particular small K values lead to a decreased classification performance, e.g., a $\mathcal{K}$-OMM-tree classifier with $K = 3$ model states yields an error rate value of ≈ 0.21.

Figure 8.6 shows graphical representations for one branch of the tree associated to class *circle*. The branch reaches from the root node (sub-figure (a)) to an actual gesture in a leaf (sub-figure (f)), i.e., the first 5 plots show OMM prototypes that represent the tree's nodes and the last plot illustrates a *circle* gesture that was used for training of the $\mathcal{K}$-OMM-tree. These plot clearly indicate that $\mathcal{K}$-OMM-trees are able to hierarchical organize gesture trajectories whereby higher levels are associated to more abstract gestures and lower nodes are connected to more concrete gesture performances.

As elements of the genuine gesture space, these prototypes could be easily used for reproduction purposes as described in Section 7.3.

According to the major research questions that we introduced in Chapter 2, these results indicate that a hierarchical time series representation as provided by $\mathcal{K}$-OMM-trees can dramatically improve the recognition performance **(Q1)** of unseen observations. As already noted in previous experiments (cf. Section 8.3), the learned hierarchy seems to reflect an inherent data structure **(Q10)** that spans from abstract data characterizations in nodes close to the root node to concrete manifestation at the bottom. Even though we did not perform the learned OMM gesture prototypes by means of the virtual agent VINCE, a visual inspection indicates that the prototypes would most likely provide good reproduction capabilities **(Q6)**.

8.5. Summary

The aim of this chapter was to investigate if OMMs are able to organize complex behavior patterns according to an inherent data structure. For this, we introduced two clustering algorithms for time series data based on OMMs (**Section 8.1**). The $\mathcal{K}$-OMMs approach is an adapted $\mathcal{K}$-means algorithm, where the cluster centroids are represented as OMMs and the negative production likelihood is used as a distance function. Further, we introduced an initialization strategy to enable compact initial cluster centroids and to avoid suboptimal solutions due to local minima. In addition, the $\mathcal{K}$-OMM-trees algorithm is a hybrid of $\mathcal{K}$-OMMs and $\mathcal{K}$-trees clustering where the tree's nodes are OMMs and its leafs hold time series observations.

We conducted a comprehensive study on 20 benchmark data sets where we compared $\mathcal{K}$-OMMs to $\mathcal{K}$-means with Euclidean distance function in **Section 8.2**. In addition, we evaluated both models with randomized initialization and compact cluster initialization. In terms of clustering purity, we found that the $\mathcal{K}$-OMMs approach reaches a higher degree of purity than $\mathcal{K}$-means for the majority of data sets, independently of the initialization strategy. Additional results show that the compact initialization could only improve the performance for few data sets. Last, the learned sequences of model parameters of the cluster centroid OMMs provide prototypes for the associated clusters and, thus, might raise further insights into a given data set.

To evaluate the hierarchical clustering approach of $\mathcal{K}$-OMM-trees, we tested the proposed algorithms in comparison to $\mathcal{K}$-HMMs on interaction time series data sets from the UCI repository related to character trajectories and sign language signs (**Section 8.3**). The results from our experiments show that all evaluated methods are able to extract clusters in interaction time series data sets with high purity. In contrast, the tested methods differ substantially in their computational running times. $\mathcal{K}$-OMMs require up to ≈ 9 times more running time, while $\mathcal{K}$-HMMs demand ≈ 60 times more computational power as compared to $\mathcal{K}$-OMM-trees. This behavior depends on the chosen hyperparameters, in particular the number of model states K and the $\mathcal{K}$-OMM-tree's order m influence the computational demands.

In **Section 8.4** we evaluated hierarchical $\mathcal{K}$-OMM-trees for recognition and reproduction of gestures. The results from these classification experiments show that $\mathcal{K}$-OMM-trees are able to recognize gesture data with substantial higher accuracy than traditional classifiers. In addition, the learned models could be directly interpreted as data prototypes. These findings indicate that the combination of improved accuracy and hierarchical data representations and prototype abstractions make $\mathcal{K}$-OMM-trees a well suited method for interactive systems. In terms of the uncovered hierarchical structures, higher levels in the tree are associated with more general gesture properties, whereas nodes closer to the leaves represent more specific gesture features. It is the extraction of this structure that enables better learning of instances of interactive behavior.

According to this thesis' **major research questions** as outlined in Section 2.1, we found evidence that OMM-based clustering algorithms are able to organize interaction time series with high accuracy. $\mathcal{K}$-OMMs provide a flat partitioning approach that can uncover data clusters **(Q9)** and provides interpretable prototypes for these clusters **(Q6)**. In addition, $\mathcal{K}$-OMM-trees are able to reveal hierarchical structures **(Q10)** in interaction data sets. Influenced by selected hyperparameters **(Q7)**, the hierarchical $\mathcal{K}$-OMM-trees requires dramatically reduced running times, which further indicates low latency capabilities of OMMs **(Q2)**. In addition, $\mathcal{K}$-OMM-trees can substantially improve the recognition performance **(Q1)** of unseen observations, and the learned hierarchy reflects an inherent data structure that spans from abstract data characterizations in nodes close to the root node to more specific manifestation at the bottom **(Q6)**.

9. Conclusion

The research objectives of this thesis were (i) to develop algorithms for machine learning of time series data that allow recognition, reproduction, and organization of observed interaction data and (ii) to empirically evaluate these algorithms with data sets from various interaction scenarios. Time series data are ubiquitous in interaction scenarios and arise in a variety of ways: e.g., gestures, speech, or sensor data streams from a smart room etc. happens in time and, thus, can be naturally represented as time series.

We addressed three prerequisites for an interactive system that is to participate in interaction environments:

- **Recognition** of human behavior,
- **Reproduction** of these behavior observations, and
- **Organization** of the observations according to an inherent data structure.

Thereby, we coped with different technical challenges: (i) fast response times to enable seaming-less interaction experiences, (ii) incremental evaluation of observations while they are performed, (iii) adaptability to particular users or environments, (iv) robustness towards incomplete observations, and (v) an algorithmic solution that integrates all of the above-mentioned prerequisites — recognition, reproduction, and organization — in only one model.

9.1. Contributions: Algorithms

Based on previous work on Ordered Means Models, the algorithms presented in this thesis allow classification, incremental classification, adaptive online training, unsupervised clustering, as well as hierarchical clustering of time series observations.

The here presented incremental classification technique empowers to obtain classification hypothesizes while a time series is observed. Thus, the classification decision is updated each time new information is available, i.e., when new samples arrive. In consequence, a classification decision can be made with specific required certainties as soon as possible, or wait for and process further evidence otherwise. The proposed online training scheme allows incremental training of OMMs observation by observation. Such a training procedure enables the ability to learn a behavior starting already with the first presentation, and to adjust the learned model incrementally when observing further instances of this behavior.

For organization of time series data, we introduced $\mathcal{K}$-OMMs. $\mathcal{K}$-OMMs are similar to classical $\mathcal{K}$-means clustering and allow discovering of time series clusters as well as cluster prototypes in an unsupervised manner. Last but importantly, we presented $\mathcal{K}$-OMM-trees, a tree-based approach to unsupervised hierarchical organization of time series data. $\mathcal{K}$-OMM-trees are height balanced clustering trees that combine classical $\mathcal{K}$-means and B+-trees algorithms with OMMs to clustering trees for time series. Here, higher levels in the tree are associated with more general prototype properties, whereas nodes closer to the leaves represent prototypes that are more specific.

9.2. Contributions: Empirical Data Analysis

In terms of OMMs' recognition capabilities we found that OMMs are well suited to process data from interaction scenarios. We successfully applied OMM-based classifiers to various interaction data sets and in different interaction scenarios. OMMs were able to recognize handwriting, speech, and sign language signs. We also succeeded in using OMMs to recognize conversational head gestures and finger-pressure patterns in musical instruments.

According to the robustness properties of OMMs in terms of incomplete observations, we found that OMMs still perform well with artificially fragmented data. These findings were established by experiments with simulation data as well as 20 benchmark data sets. In comparison to HMMs with different topologies the simplification provided by OMMs significantly increases the robustness with regard to classification of incomplete data. In a conversational head gesture scenario, OMMs were able to recognize head gestures with comparatively low error rates even if only 10% of the genuine head gesture was performed. This indicates that head gestures are still well separable if only partial observations are available. All together, the omission of transition probabilities represents a valuable alternative to standard HMMs when robust recognition is crucial.

Although OMMs provide high model flexibility and consistent classification accuracy, we found that the computational costs of OMMs are even lower than the costs of HMMs with linear model topology. In comparison to equally flexible left-to-right HMMs, the considerable speed-up is still practically relevant for interaction scenarios.

Results from experiments on real-world interaction data sets indicate that OMMs are well suited to process interaction data, and in contrast to highly accurate kernel machines, the computational efficiency of OMM classifiers is substantially lower and, thus, enable real-time recognition in interaction scenarios. Additional results from gesture recognition experiments reveal that OMMs are able to learn time series gestures even if only few observations are available. Here, OMMs reach competitive accuracy with few model states what leads to considerably decreasing computational demands and, thus, also ensures low latency capability.

To see whether OMMs are able to incrementally recognize interaction time series and can adapt to particular users, we realized the Rock-Paper-Scissors game as an application and test bed scenario in which the virtual agent VINCE engages in fast, real-life gesture-based interaction with human users. This scenario directly taps into the abilities of the model for incremental recognition and adaption. The results we obtained in an evaluation study indicate that OMMs can meet the before-mentioned requirements: incremental recognition with high accuracies, as well as the ability to adapt to particular users via online training.

In terms of reproduction of interaction data, we investigated OMMs' prototype property. In evaluation of mouse gestures and hand-arm gestures, we found that OMMs are feasible to reproduce interaction data. Thereby, the deviation parameter σ of OMMs controls the smoothness of the prototype with higher values of σ yielding smoother prototypes. The number of model states K governs the number of samples of the prototype, i.e., the pace or the sample frequency. In comparison to reproduction capabilities of similar HMMs, the results indicate that OMMs are able to reveal the relevant information quicker, however, depending on the chosen model topology, HMMs were also capable to provide a feasible prototype. The learned prototype representations for a hand arm gesture data set could also be used for classification of unseen gestures with high accuracy — a strong indicator that OMMs provide an integrated approach to reproduction and recognition of interaction data.

According to OMMs' organizational features, we found that OMM-based clustering algorithms are able to partition a given interaction time series data set, as well as uncover hidden hierarchical structures, both in an unsupervised manner. A study on benchmark data revealed that for the majority of data sets, $\mathcal{K}$-OMMs reach a higher degree of purity as compared to standard $\mathcal{K}$-means. Further results indicate that these findings are independent of the chosen cluster initialization. As an additional outcome of these experiments, we found that the learned cluster centroids provide interpretable prototypes for the associated time series clusters. This indicates that OMMs are able to integrate organization and reproduction properties in a single model.

Results from experiments on interaction time series data revealed that both — $\mathcal{K}$-OMMs and $\mathcal{K}$-OMM-trees — are able to extract data clusters from such data with high purity. However, the tested methods differ substantially in their computational costs. Influenced by selected hyperparameters, hierarchical $\mathcal{K}$-OMM-trees require dramatically reduced running times in comparison to $\mathcal{K}$-OMMs and a similar approach based on HMMs. In terms of prototype extraction, the learned hierarchy of $\mathcal{K}$-OMM-trees reflects an inherent data structure that spans from abstract data characterizations in nodes close to the root node to more specific manifestation at the bottom. Further experiments toward gesture recognition showed that $\mathcal{K}$-OMM-trees are able to substantially improve the recognition performance of unseen gestures.

9.3. Future Work

The work and results presented in this thesis prepare the way for future work in various research directions, both, driven by algorithms and applications.

In our current definition of OMMs, the emission densities are represented as Gaussians whose standard deviation parameter is used as a global hyperparameter. Here, using alternative distribution models could allow a more suitable representation of time series or increased robustness towards noisy observations. Appropriate choices would be, e.g., mixture-of-Gaussians or double exponential distributions. In addition, estimating all parameters of the emission densities in training might yield adequate data representations with reduced costs for model selection.

Since OMMs also provide a probabilistic similarity function for two time series, they could also be employed as time series density kernels in kernel density estimators and related methods. In such an approach, OMMs' standard deviation parameter would be used as a kernel bandwidth parameter. In this context, sparse, discriminative classification approaches based on kernel density estimation such as maximum contrast classifiers [Meinicke et al., 2002] might further improve OMMs' recognition performance. In order to preserve OMMs' excellent response times and nevertheless allow discriminative classification, the objective function of OMMs could be altered to not only maximize the class-specific likelihoods, but also minimize the likelihoods related to observations from other classes.

According to reproduction, promising research directions are OMMs' generation properties. In contrast to reproduction by means of prototypes, generation would yield greater variation in the reproduced time series. Here, it could be beneficial to limit the path selection in a generating process to paths that are close to the diagonal in the dynamic programming — similar to pruning for DTW.

In terms of organization of time series data, straightforward extensions of $\mathcal{K}$-OMMs are self-organizing maps (SOM) based on OMMs that would allow processing of time series. In such an approach, SOM neurons would be represented by OMMs to which observations are assigned according to their production likelihoods. But also other manifold estimators for time series are possible: the time series distance introduced in this thesis could be applied to isomap — as well as all isometric mapping learners based on pairwise distances — to discover manifolds in time series space.

In addition to algorithmic extensions, the work of this thesis can contribute to countless application in various scenarios. Among others, the proposed algorithms for organization of interaction time series enable activity clustering of, e.g., sensor readings in smart phones to predict potential medical or emotional states of the user; OMM-based classifiers could be used in multimodal dialog situations with a sociable agent in order to allow adaption to adequate social categories; last but not least, the prototype property of OMMs directly allows imitation learning of gestures and mimics in robotics and virtual agent research. However, not only interactive systems can benefit from OMMs: time series data arises in various ways, and robust, rapid, and reliable algorithms to process this data enable exciting applications in many other domains.

A. Expectation Maximization Algorithm

If we understand an OMM as a mixture of M^i components — here: $p(O^i|\mathbf{q},\Omega)$ — with constant weights $w_m = \frac{1}{M^i}$, we can maximize $\mathcal{L}$ by iterating

expectation step:

$$\begin{aligned}
h^i_{\mathbf{q}} &= P(\mathbf{q}|O^i,\Omega) \\
&= \frac{p(O^i|\mathbf{q},\Omega)P(\mathbf{q}|\Omega)}{\sum\limits_{\mathbf{q}'\in Q^i} p(O^i|\mathbf{q}',\Omega)P(\mathbf{q}'|\Omega)} = \frac{p(O^i|\mathbf{q},\Omega)}{\sum\limits_{\mathbf{q}'\in Q^i} p(O^i|\mathbf{q}',\Omega)} \\
&= \frac{\prod\limits_{t=1}^{T^i} b_{q_t}(\mathbf{o}_t)}{\sum\limits_{\mathbf{q}'\in Q^i}\prod\limits_{t=1}^{T^i} b_{q'_t}(\mathbf{o}_t)} = \frac{\left(\frac{1}{(2\pi\sigma^2)^{\frac{d}{2}}}\right)^{T^i}\prod\limits_{t=1}^{T^i} \tilde{b}_{q_t}(\mathbf{o}_t)}{\left(\frac{1}{(2\pi\sigma^2)^{\frac{d}{2}}}\right)^{T^i}\sum\limits_{\mathbf{q}'\in Q^i}\prod\limits_{t=1}^{T^i} \tilde{b}_{q'_t}(\mathbf{o}_t)} \qquad \text{(A.1)}
\end{aligned}$$

maximization step:

$$\begin{aligned}
\hat{\Omega} = [\hat{\boldsymbol{\mu}}_1,\ldots,\hat{\boldsymbol{\mu}}_K] &= \arg\max_{\Omega}\sum_{i=1}^{N}\sum_{\mathbf{q}\in Q^i} h^i_{\mathbf{q}} \ln p(O^i|\mathbf{q},\Omega) \\
&= \arg\max_{\Omega}\sum_{i=1}^{N}\sum_{\mathbf{q}\in Q^i} h^i_{\mathbf{q}} \left(\sum_{t=1}^{T^i}\frac{-\|\mathbf{o}^i_t-\boldsymbol{\mu}_{q_t}\|^2}{2\sigma^2} - \frac{T^i d}{2}\ln(2\pi\sigma^2)\right) \\
&= \arg\min_{\Omega}\sum_{i=1}^{N}\sum_{\mathbf{q}\in Q^i} h^i_{\mathbf{q}} \sum_{t=1}^{T^i}\|\mathbf{o}^i_t-\boldsymbol{\mu}_{q_t}\|^2. \qquad \text{(A.2)}
\end{aligned}$$

Both terms $\frac{T^i d}{2}\ln(2\pi\sigma^2)$ und $2\sigma^2$ are independent of Ω according a maximized $\mathcal{L}$. And the normalization in the expectation steps leads to $\sum_{\mathbf{q}\in Q^i} h^i_{\mathbf{q}} = 1$.

The summation of all path probabilities $h^i_{\mathbf{q}}$ that assign $\mathbf{o}^i_t$ to $\boldsymbol{\mu}_k$ defines the *responsibilities*:

$$\begin{aligned}
r^i_{k,t} &\equiv \frac{P(\mathbf{S}^i, q_t = k|\Omega)}{P(\mathbf{S}^i|\Omega)} \\
&= \sum_{\substack{\mathbf{q}\in Q: \\ q_t=k}} h^i_{\mathbf{q}} = \frac{\sum\limits_{\substack{\mathbf{q}\in Q: \\ q_t=k}}\prod\limits_{t=1}^{T^i} b_{q_t}(\mathbf{o}^i_t)}{\sum\limits_{\mathbf{q}'\in Q^i}\prod\limits_{t=1}^{T^i} b_{q'_t}(\mathbf{o}^i_t)} \qquad \text{(A.3)} \\
\Rightarrow \sum_{k=1}^{K} r^i_{k,t} &= \sum_{k=1}^{K}\sum_{\substack{\mathbf{q}\in Q^i: \\ q_t=k}} h^i_{\mathbf{q}} \overset{!}{=} 1. \qquad \text{(A.4)}
\end{aligned}$$

These responsibilities $r^i_{k,t}$ are used in the maximization step to estimate updated prototypes $\Omega = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K]$:

$$\Omega = \arg\min_{\Omega} E_{Reg} = \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{t=1}^{T^i} r^i_{k,t} \| \mathbf{o}^i_t - \boldsymbol{\mu}_k \|^2$$

with

$$\begin{aligned} \nabla_{\boldsymbol{\mu}_k} E_{Reg} &= \sum_{i=1}^{N} \sum_{t=1}^{T^i} r^i_{k,t} \left(-2\mathbf{o}^i_t + 2\boldsymbol{\mu}_k \right) \overset{!}{=} \mathbf{0} \\ \Rightarrow \boldsymbol{\mu}_k &= \frac{\sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T^i} r^i_{k,t} \cdot \mathbf{o}^i_t}{\sum\limits_{i=1}^{N} \sum\limits_{t=1}^{T^i} r^i_{k,t}}. \end{aligned} \tag{A.5}$$

These steps are repeated until the error function converges.

B. pyOMM: An OMM Package for the Python Programming Language

PyOMM is an implementation of the OMM algorithms in the Python programming language. The package provides algorithms for training of OMMs as well as classification, clustering, and model selection solutions for time series based on OMMs. In addition, necessary data structures to represent multivariate time series and time series data sets are available.

Copyright 2013: Ulf Großekathöfer, ugrossek@gmail.com

B.1. Installation Instructions

The pyOMM package requires scipy. If you do not have scipy installed on your system, please go to http://www.scipy.org/, download, and install it. Even though most parts of pyOMM is written in the Python programming language, time critical parts such as the dynamic programming are implemented in the C programming language. In order to access C code from python, scipy.weave is used. Please make sure that you have scipy.weave installed on your system. Due to a bug in scipy.weave, pyOMM requires a gnu C compiler version before gcc-4.2. PyOMM was written for Python version 2.5 or higher, however, it might be compatible to previous versions of Python. Please note that pyOMM does not work with Python 3.0.

For installation unpack pyomm-¡version¿.tar.gz, change to the archive directory, and call the installation script. For example, if you are using Linux the commands are:

```
> tar xvfz pyomm-<version>.tar.gz
> cd pyomm-<version>
> python setup.py install
```

To obtain a complete list of installation options, call:

```
> python setup.py help
```

B.2. How to use pyOMM

In order to use pyOMM, you first need to import the module `omm` in your python program. You then can use the provided functions and classes. From a user point of view, the most important functions are:

- `omm.classifier(flavour="OMM", data_set, K, param)` initializes a classifier.

- `omm.cluster(flavour, K, param, C, sequence_or_data_set, init)` for initialization of a clustering of time series data.
- `omm.omm(sequence_set, K, param)` initializes an OMM according to the provided parameters.

In order to select a specific model for classification or clustering, the arguemnt `flavor` can be used; `flavor` understands the following values: `"omm"`: for use of OMMs, `"ommonline"`: to use the incremental variant of OMMs, `"lrhmm"`: a similar HMM with left-to-right model topology, `"linearhmm"`: to specify an HMM with linear model topology, or `"brownhmm"`: an HMMwith linear model topology and predefined transition probabilities.

Important Data Structures

Time series are represented as numpy arrays in the pyOMM package. The data representation comprises the following classes:

- `class Sequence` stores a single time series or sequence observation in a numpy array.
- `class SequenceSet` is a container class for sequences. Note that a `SequenceSet` is not actually a set but a list.
- `class Class` is a `SequenceSet` with a label.
- `class DataSet` is a list of `Class` objects.

pyOMM provides implementations for various OMM and HMM variants that as used in this thesis:

- `class OMM` represents an Ordered Means Model.
- `class OMMAll` is class related to a probabilistic representation of Ordered Means Model that implements the log-space training approach.
- `class OMMBest` is a deterministic OMM variant (with $\sigma \to 0$) and Viterbi training.
- `class OMMAllScale` is a probabilistic Ordered Means Model with a training that is related to scaling of probabilities.
- `class OMMAnytime` implements the incremental evaluation scheme for sample by sample evaluation.
- `class HMM` is a Hidden Markov Model with left-to-right topology,
- `class HMMLinear` provides an HMM with linear model topology,
- `class HMMBrown` is related to an HMM implementation with pre-defined transition probabilities that do not have to be estimated from the observations.

C. pyKTree: A Python Implementation of the $\mathcal{K}$-tree Algorithm

PyKTree provides a $\mathcal{K}$-tree implementation in the Python programming language. $\mathcal{K}$-trees are a scalable approach to clustering by combining the B+-tree and $\mathcal{K}$-means algorithms. Clustering can be used to solve problems in signal processing, machine learning, and other contexts.

C.1. Installation Instructions

The pyktree package requires scipy. If you do not have scipy installed on your system, please go to http://www.scipy.org/, download, and install it. Additionally, you will need Python version 2.5 or higher. Please note that this $\mathcal{K}$-tree implementation does not work with Python 3.0 yet.

For installation unpack pyktree-¡version¿.tar.gz, change to the archive directory, and call the installation script. For example, if you are using Linux the commands are:

```
> tar xvfz pyktree-<version>.tar.gz
> cd pyktree-<version>
> python setup.py install
```

To obtain a complete list of installation options, call:

```
> python setup.py help
```

C.2. How to use pyKTrees

There are two ways of using pyktree: first, as a Python library, or, second, as command line tools.

Library use

Import the module `ktree` in your python program and use the provided functions and classes. The module provides a function `ktree.ktree(order=5, data_set)` that generates a $\mathcal{K}$-tree from a `data_set` of order `order`.

Please see `example/ktree_example.py` for example code.

Command line use

pyktree comes with three command line scripts: `ktmk`, `ktnn`, `ktpr`. ktmk builds a K-Tree for a data set and saves it to a file; ktnn searches the nearest neighbors in a K-Tree for a set of unseen examples, while ktpr prints an ASCII representation of a K-tree to stdout. Please note, this representation can get very large, even for (relatively) small K-Trees. It is not recommended to use ktpr for K-Trees with more than 1000 examples.

A typical use case would look like this:

```
> # Build a K-Tree for a data set:
> ktmk --data-set data.txt --order 6 --k-tree ktree.ktf -m -r

> # Print the K-Tree:
> ktpr ktree.ktf

> # Find nearest neighbors in K-Tree for unseen examples:
> ktnn --k-tree ktree.ktf --data-set unseen.txt --outfile nn.txt
```

Bibliography

A. Akl and S. Valaee. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 2270–2273. IEEE, 2010.

J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic. Discovering clusters in motion time-series data. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–375. IEEE, 2003.

E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval*, 12(4):461–486, 2009.

M. R. Anderberg. Cluster analysis for applications. Technical report, DTIC Document, 1973.

S. Arora, D. Bhattacharjee, M. Nasipuri, D. Basu, and M. Kundu. Recognition of non-compound handwritten devnagari characters using a combination of mlp and minimum edit distance. *arXiv preprint arXiv:1006.5908*, 2010.

T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. *International Journal of Humanoid Robotics*, 5(02): 183–202, 2008.

S. Asthana, F. Haneef, and R. Bhujade. Handwritten multiscript numeral recognition using artificial neural networks. *International Journal of Soft Computing and Engineering*, 1 (1):1–5, 2011.

A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 79–85. Association for Computational Linguistics, 1998.

C. Bahlmann, B. Haasdonk, and H. Burkhardt. Online handwriting recognition with support vector machines-a kernel approach. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, pages 49–54. IEEE, 2002.

L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Amer. Math. Soc*, 73(3):360–363, 1967.

L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

L. E. Baum and G. R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

S. Bercu and G. Lorette. On-line handwritten word recognition: an approach based on hidden markov models. In *Proceeding Third Int. Workshop on Frontiers in Handwriting Recognition*, pages 385–390, 1993.

K. Bergmann and S. Kopp. GNetIc – using bayesian decision networks for iconic gesture generation. In *Proceedings of the 9th Conference on Intelligent Virtual Agents*, pages 76–89. Springer, 2009.

P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

Z. Bin, L. Yong, and X. Shao-Wei. Support vector machine and its application in handwritten numeral recognition. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 720–723. IEEE, 2000.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, Aug. 2006. ISBN 0387310738.

A. Black and P. Taylor. Automatically clustering similar units for unit selection in speech synthesis. 1997.

A. Black, H. Zen, and K. Tokuda. Statistical parametric speech synthesis. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–1229. IEEE, 2007.

M. Black and A. Jepson. A probabilistic framework for matching temporal trajectories: Condensation-based recognition of gestures and expressions. *Computer Vision—ECCV'98*, pages 909–924, 1998.

A. Bobick and A. Wilson. A state-based approach to the representation and recognition of gesture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(12): 1325–1337, 1997.

L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 77–82. IEEE, 1994.

H. Bourlard and N. Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer, 1994.

R. Bowden, A. Zisserman, T. Kadir, and M. Brady. Vision based interpretation of natural sign languages. 2003.

R. Bowden, D. Windridge, T. Kadir, A. Zisserman, and M. Brady. A linguistic feature vector for the visual interpretation of sign language. *Computer Vision-ECCV 2004*, pages 390–401, 2004.

C. Breazeal. Toward sociable robots. *Robotics and Autonomous Systems*, 42(3):167–175, 2003.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

S. Calinon and A. Billard. Stochastic gesture production and recognition model for a humanoid robot. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2769–2774. IEEE, 2004.

S. Calinon and A. Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the 22nd international conference on Machine learning*, pages 105–112. ACM, 2005.

S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 255–262. ACM, 2007.

J. Campbell. *Grammatical man: Information, entropy, language, and life.* Simon and Schuster New York, 1982.

J. Cassell, C. Pelachaud, N. Badler, M. Steedman, B. Achorn, T. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 413–420. ACM New York, NY, USA, 1994.

C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 792–803. VLDB Endowment, 2004.

L. Chen, M. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

V. S. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods.* John Wiley & Sons, Inc., New York, NY, USA, 1998. ISBN 0471154938.

S. Chiba and H. Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43, 1978.

P. Clarkson and P. Moreno. On the use of support vector machines for phonetic classification. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, pages 585–588 vol.2. IEEE, 1999.

D. Comer. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)*, 11(2):121–137, 1979.

A. Conkie. A robust unit selection system for speech synthesis. In *Joint Meeting of ASA/EAA/DAGA, Berlin, Germany, March 1999.* Citeseer, 1999.

M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui. A Kernel for Time Series Based on Global Alignments. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, pages II–413–II–416. IEEE, 2007.

J. Davis and M. Shah. Visual gesture recognition. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 141, pages 101–106. IET, 1994.

C. De Vries. Application of k-tree to document clustering. Master's thesis, 2010.

C. M. De Vries and S. Geva. Document Clustering with K-tree. *Advances in Focused Retrieval: 7th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2008, Schloss Dagstuhl, Germany*, pages 420–431, 2009a.

C. M. De Vries and S. Geva. K-tree: large scale document clustering. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 718–719. ACM, 2009b.

C. M. De Vries, L. De Vine, and S. Geva. Random Indexing K-tree. *ADCS 2009: Australian Document Computing Symposium 2009, Sydney, Australia*, pages 43–50, 2009.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the {EM} Algorithm. *J. Roy. Stat. Soc. B*, 39:1–38, 1977.

H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endowment Archive*, 1(2):1542–1552, 2008.

A. Dix. *Human-computer interaction*. Prentice hall, 2004.

B. Dom. An information-theoretic external cluster-validity measure. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 137–145. Morgan Kaufmann Publishers Inc., 2002.

S. R. Eddy. Profile hidden Markov models (review). *Bioinformatics*, 14(9):755–763, 1998.

A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Suen. An hmm-based approach for off-line unconstrained handwritten word modeling and recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(8):752–760, 1999.

T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

J. D. Ferguson. Variable duration models for speech. In *Proceedings of the Symposium on the Application of hidden Markov models to Text and Speech*, volume 1, pages 143–179, 1980.

A. Finke, A. Lenhardt, and H. Ritter. The mindgame: a p300-based brain–computer interface game. *Neural Networks*, 22(9):1329–1333, 2009.

G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

A. Frank and A. Asuncion. {UCI} machine learning repository, 2010.

A. Ganapathiraju, J. Hamaker, and J. Picone. Applications of support vector machines to speech recognition. *Signal Processing, IEEE Transactions on*, 52(8):2348–2355, 2004.

D. Gavrila, L. Davis, et al. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *International workshop on automatic face-and gesture-recognition*, pages 272–277. Citeseer, 1995.

S. Geva. K-tree: a height balanced tree structured vector quantizer. In *NEURAL NETWORKS SIGNAL PROCESS PROC IEEE*, volume 1, pages 271–280, 2000.

A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

U. Großekathöfer and T. Lingner. Neue Ansätze zum maschinellen Lernen von Alignments. Diplomarbeit, Bielefeld University, Neuroinformatics Group, 2005.

U. Großekathöfer, A. Barchunova, R. Haschke, T. Hermann, M. Franzius, and H. Ritter. Learning of Object Manipulation Operations from Continuous Multimodal Input. In *IEEE/RAS International Conference on Humanoid Robots 2011*, 2011.

U. Großekathöfer, S. Geva, T. Hermann, and S. Kopp. Learning hierarchical prototypes of motion time series for interactive systems. In *ECAI Workshop on Machine Learning for Interactive Systems (MLIS)*, 2012.

U. Großekathöfer, A. Sadeghipour, T. Lingner, P. Meinicke, T. Hermann, and S. Kopp. Low Latency Recognition and Reproduction of Natural Gesture Trajectories. In *ICPRAM (Int.Conf. on Pattern Recognition Applications and Methods)*, 2012.

U. Großekathöfer, N.-C. Wöhler, T. Hermann, and S. Kopp. On-the-fly behavior coordination for interactive virtual agents: a model for learning, recognizing and reproducing hand-arm gestures online. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1177–1178. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

T. Grosshauser. Low force pressure measurement: Pressure sensor matrices for gesture analysis, stiffness recognition and augmented instruments. In *8th International Conference on New Interfaces for Musical Expression NIME08*. Citeseer, 2008.

T. Grosshauser, U. Großekathöfer, and T. Hermann. New sensors and pattern recognition techniques for string instruments. In *NIME 2010: Proceedings of the 2010 conference on New interfaces for musical expression, 15-18th June 2010, Proceedings*, 2010.

S. Gudmundsson, T. P. Runarsson, and S. Sigurdsson. Support vector machines and dynamic time warping for time series. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, (x):2772–2776, June 2008.

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

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

J. Hamaker, J. Picone, and A. Ganapathiraju. A sparse modeling approach to speech recognition based on relevance vector machines. In *Proceedings of the International Conference of Spoken Language Processing*, volume 2, pages 1001–1004, 2002.

G. Heidemann, H. Bekel, I. Bax, and A. Saalbach. Hand gesture recognition: self-organising maps as a graphical user interface for the partitioning of large training data sets. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 4, pages 487–490. IEEE, 2004.

B. Hellwig and D. Uytvanck. EUDICO Linguistic Annotator (ELAN) Version 2.0. 2 manual. *Nijmegen-NL, Max Planck Institute for Psycholinguistics*, 2004.

P. Hong, M. Turk, and T. Huang. Gesture modeling and recognition using finite state machines. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 410–415. IEEE, 2000.

J. Hu, M. Brown, and W. Turin. Hmm based online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(10):1039–1045, 1996.

A. Hunt and A. Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 373–376. IEEE, 1996.

T. Inamura, H. Tanie, and Y. Nakamura. Keyframe compression and decompression for time series data based on the continuous hidden markov model. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1487–1492. IEEE, 2003.

T. Jaakkola, M. Diekhaus, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

J. Jay, J. Kim, and H. Jin. Data-driven design of hmm topology for online handwriting recognition. *International journal of pattern recognition and artificial intelligence*, 15 (01):107–121, 2001.

T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *The Journal of Machine Learning Research*, 5:819–844, 2004.

T. Jebara, Y. Song, and K. Thadani. Spectral clustering and embedding with hidden markov models. *Machine Learning: ECML 2007*, pages 164–175, 2007.

M. T. Johnson. Capacity and complexity of HMM duration modeling techniques. *IEEE Signal Processing Letters*, 12(5):407–410, 2005.

B. Juang and S. Katagiri. Discriminative learning for minimum error classification [pattern recognition]. *Signal Processing, IEEE Transactions on*, 40(12):3043–3054, 1992.

B. Juang and L. Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

B. Juang, W. Hou, and C. Lee. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 5(3):257–265, 1997.

M. W. Kadous. *Temporal classification: Extending the classification paradigm to multivariate time series.* PhD thesis, Citeseer, 2002.

W. Kadous. Grasp: Recognition of australian sign language using instrumented gloves. Technical report, 1995.

M. Kaper, P. Meinicke, U. Grossekathoefer, T. Lingner, and H. Ritter. Bci competition 2003-data set iib: Support vector machines for the p300 speller paradigm. *Biomedical Engineering, IEEE Transactions on*, 51(6):1073–1076, 2004.

G. Karypis. CLUTO - a clustering toolkit. Technical Report #02-017, Nov. 2003.

S. Katagiri, B. Juang, and C. Lee. Pattern recognition using a family of design algorithms based upon the generalized probabilistic descent method. *Proceedings of the IEEE*, 86 (11):2345–2373, 1998.

V. Kellokumpu, M. Pietikäinen, and J. Heikkilä. Human activity recognition using sequences of postures. In *Proceedings of the IAPR conference on machine vision applications (MVA 2005), Tsukuba Science City, Japan*, pages 570–573, 2005.

E. Keogh and M. Pazzani. Scaling up dynamic time warping to massive datasets. *Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, May 2004. ISSN 0219-1377.

E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/\textasciitilde eamonn, 2006.

S. Kopp and I. Wachsmuth. Synthesizing multimodal utterances for conversational agents. *Computer Animation and Virtual Worlds*, 15(1):39–52, 2004. ISSN 1546-4261.

G. Kramer. *Auditory display: Sonification, audification, and auditory interfaces.* Addison-Wesley, 1994.

F. Kurniawan, A. Rehman, D. Mohamad, and S. Shamsudin. Self organizing features map with improved segmentation to identify touching of adjacent characters in handwritten words. In *Hybrid Intelligent Systems, 2009. HIS'09. Ninth International Conference on*, volume 1, pages 475–480. IEEE, 2009.

C. Kwok, D. Fox, and M. Meila. Real-time particle filters. *Proceedings of the IEEE*, 92(3): 469–484, 2004.

J. Kwon and F. C. Park. Natural Movement Generation Using Hidden Markov Models and Principal Components. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(5):1184–1194, 2008.

P. Langley, G. Drastal, R. B. Rao, and R. Greiner. Elements of machine learning, 1996.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

S. E. Levinson. Continuously variable duration hidden Markov models for automatic speech recognition. *Computer Speech & Language*, 1(1):29–45, 1986.

P. Liang and D. Klein. Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 611–619. Association for Computational Linguistics, 2009.

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

C. Liu, H. Sako, and H. Fujisawa. Performance evaluation of pattern classifiers for handwritten character recognition. *International Journal on Document Analysis and Recognition*, 4(3):191–204, 2002.

B. Logan et al. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, volume 28, page 5, 2000.

H.-L. Lou. Implementing the Viterbi algorithm. *Signal Processing Magazine, IEEE*, 12(5): 42–52, Sept. 1995.

J. B. MacQueen and Others. Some methods for classification and analysis of multivariate observations, 1966.

U. Marti and H. Bunke. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):65–90, 2001.

S. Maskell and N. Gordon. A tutorial on particle filters for on-line nonlinear/non-gaussian bayesian tracking. In *Target Tracking: Algorithms and Applications (Ref. No. 2001/174), IEE*, pages 2–1. IET, 2001.

E. McDermott, T. Hazen, J. Le Roux, A. Nakamura, and S. Katagiri. Discriminative training for large-vocabulary speech recognition using minimum classification error. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(1):203–223, 2007.

D. McNeill. *Hand and mind: What gestures reveal about thought.* University of Chicago Press, 1992.

M. Meila. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, pages 577–584. ACM, 2005.

P. Meinicke, T. Twellmann, and H. Ritter. Discriminative densities from maximum contrast estimation. *Advances in Neural Information Processing Systems*, 15:985–992, 2002.

L. P. Morency, A. Rahimi, N. Checka, and T. Darrell. Fast stereo-based head tracking for interactive environments. In *Fifth IEEE International Conference on Automatic Face and Gesture Recognition, 2002. Proceedings*, pages 390–395, 2002.

P. Moreno and R. Rifkin. Using the fisher kernel method for web audio classification. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, volume 6, pages 2417 –2420 vol.4, 2000.

K. G. Munhall, J. A. Jones, D. E. Callan, T. Kuratate, and E. Vatikiotis-Bateson. Head movement improves auditory speech perception. *Psychological Science*, 15(2):133–137, 2004.

B. Myers. A brief history of human-computer interaction technology. *interactions*, 5(2): 44–54, 1998.

A. Nádas, D. Nahamoo, and M. Picheny. On a model-robust training method for speech recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(9): 1432–1436, 1988.

S. Nakajima and H. Hamada. Automatic generation of synthesis units based on context oriented clustering. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 659–662. IEEE, 1988.

R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

N. J. Nilsson. *Introduction to Machine Learning (An Early Draft to a proposed textbook).* 1996.

T. Oates, L. Firoiu, and P. R. Cohen. Clustering time series with hidden markov models and dynamic time warping. In *Proceedings of the IJCAI-99 workshop on neural, symbolic and reinforcement learning methods for sequence learning*, pages 17–21. Citeseer, 1999.

B. Obermaier, C. Guger, C. Neuper, and G. Pfurtscheller. Hidden Markov models for online classification of single trial EEG data. *Pattern Recognition Letters*, 22(12):1299–1309, 2001.

J. Pearsall and P. Hanks. *The new Oxford dictionary of English.* Clarendon Press, 1998.

M. P. Perrone and S. D. Connell. K-means clustering for hidden Markov models. In *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition*, pages 229–238. Citeseer, 2000.

T. Plötz and G. Fink. Markov models for offline handwriting recognition: a survey. *International journal on document analysis and recognition*, 12(4):269–298, 2009.

F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

A. Ramamoorthy, N. Vaswani, S. Chaudhury, and S. Banerjee. Recognition of dynamic hand gestures. *Pattern Recognition*, 36(9):2069–2081, 2003.

T. M. Rath and R. Manmatha. Word Image Matching Using Dynamic Time Warping. In *CVPR (2)*, volume 2, pages 521–527. IEEE Computer Society, 2003. ISBN 0-7695-1900-8.

A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, volume 410, page 420, 2007.

M. Rosenblum, Y. Yacoob, and L. Davis. Human expression recognition from motion using a radial basis function network architecture. *Neural Networks, IEEE Transactions on*, 7 (5):1121–1138, 1996.

S. Roweis and Z. Ghahramani. A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2):305–345, 1999.

M. Russell and A. Cook. Experimental evaluation of duration modelling techniques for automatic speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'87.*, volume 12, pages 2376–2379. IEEE, 1987.

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1): 43–49, 1978.

M. Salem, S. Kopp, I. Wachsmuth, K. Rohlfing, and F. Joublin. Generation and evaluation of communicative robot gesture. *International Journal of Social Robotics*, pages 1–17, 2012.

F. Samaria and S. Young. Hmm-based architecture for face identification. *Image and vision computing*, 12(8):537–543, 1994.

M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time-delay neural networks and hidden markov models. *Machine Vision and Applications*, 8 (4):215–223, 1995.

T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14. ACM, 2008.

L. Schomaker. Using stroke-or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

H. Shimodaira, K. ichi Noma, M. Nakai, and S. Sagayama. Dynamic Time-Alignment Kernel in Support Vector Machine.

N. Smith and M. Niranjan. Data-dependent kernels in svm classification of speech patterns. In *The Proceedings of the 6(th) International Conference on Spoken Language Processing (Volume I)*, 2001.

T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

A. Smola and B. Schölkopf. *Learning with kernels.* Citeseer, 1998.

P. Smyth. Clustering sequences with hidden Markov models. *Advances in neural information processing systems*, 1997.

T. Starner and A. Pentland. Real-time american sign language recognition from video using hidden markov models. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 265–270. IEEE, 1995.

T. Starner, J. Weaver, and A. Pentland. Real-time american sign language recognition using desk and wearable computer based video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(12):1371–1375, 1998.

J. Steffen. Structured manifolds for motion production and segmentation: a structured kernel regression approach. 2010.

J. Steffen, R. Haschke, and H. Ritter. Experience-based and tactile-driven dynamic grasp control. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2938–2943. IEEE, 2007.

J. Steffen, R. Haschke, and H. Ritter. Towards dextrous manipulation using manipulation manifolds. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2738–2743. IEEE, 2008.

M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

K. Sung and T. Poggio. Learning human face detection in cluttered scenes. In *Computer Analysis of Images and Patterns*, pages 432–439. Springer, 1995.

K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1315–1318. IEEE, 2000.

K. Tokuda, H. Zen, and A. Black. An hmm-based speech synthesis system applied to english. In *Speech Synthesis, 2002. Proceedings of 2002 IEEE Workshop on*, pages 227–230. IEEE, 2002.

D. Tolani, A. Goswami, and N. I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

B. Ullmer and H. Ishii. Emerging frameworks for tangible user interfaces. *IBM systems journal*, 39(3.4):915–931, 2000.

V. N. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 2000. ISBN 0387987800.

S. V. Vaseghi. State duration modelling in hidden Markov models. *Signal processing*, 41 (1):31–41, 1995.

R. D. Vatavu, b. Pentiuc, and C. Chaillou. On Natural Gestures for Interacting in Virtual Environments. *Advances in Electrical and Computer Engineering*, 24(5), 2005.

M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

V. Vuori, J. Laaksonen, E. Oja, and J. Kangas. Experiments with adaptation strategies for a prototype-based recognition system for isolated handwritten characters. *International Journal on Document Analysis and Recognition*, 3(3):150–159, 2001.

L. Vuurpijl and L. Schomaker. Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 1, pages 387–393. IEEE, 1997.

J. Wachs, H. Stern, and Y. Edan. Cluster labeling and parameter estimation for the automated setup of a hand-gesture recognition system. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(6):932–944, 2005.

V. Wan and S. Renals. Evaluation of kernel methods for speaker verification and identification. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–669 –I–672, may 2002.

T. Wang, H. Shum, Y. Xu, and N. Zheng. Unsupervised analysis of human gestures. *Advances in Multimedia Information Processing—PCM 2001*, pages 174–181, 2001.

C. Watkins. Dynamic Alignment Kernels. *Advances in Large Margin Classifiers*, pages 39 – 50, 1999.

B. Williams, M. Toussaint, and A. Storkey. Extracting motion primitives from natural handwriting data. *Artificial Neural Networks–ICANN 2006*, pages 634–643, 2006.

N.-C. Wöhler. Maschinelles lernen von kopfgesten mit ordered means models. Bachelor's thesis, Bielefeld University, Neuroinformatics Group, 2009.

N.-C. Wöhler. Anytime klassifikation mit ordered means models am beispiel eines gesteninteraktions-spiels mit einem virtuellen agenten. Master's thesis, Bielefeld University, Neuroinformatics Group, 2012.

N.-C. Wöhler, U. Großekathöfer, A. Dierker, M. Hanheide, S. Kopp, and T. Hermann. A calibration-free head gesture recognition system with online capability. In *International Conference on Pattern Recognition*, Istanbul, Turkey, 2010.

N. Xiong and P. Svensson. Multi-sensor management for information fusion: issues and approaches. *Information fusion*, 3(2):163–186, 2002.

W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE, 1992.

H. Yan. *Paired speech and gesture generation in embodied conversational agents.* PhD thesis, Massachusetts Institute of Technology, 2000.

M. Yang and N. Ahuja. Recognizing hand gesture using motion trajectories. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

M. Yeasin and S. Chaudhuri. Visual understanding of dynamic hand gestures. *Pattern Recognition*, 33(11):1805–1817, 2000.

H. Yoon, J. Soh, Y. Bae, and H. Seung Yang. Hand gesture recognition using combined features of location, angle and velocity. *Pattern Recognition*, 34(7):1491–1501, 2001.

T. Yoshimura, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura. Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis. 1999.

Y. Zhao and G. Karypis. Criterion functions for document clustering: Experiments and analysis. *Machine Learning*, 2001.