Feature-based Time Series Analytics

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Inf. Lars Kegel
geboren am 16. Juni 1988 in Dresden

Gutachter:	**Prof. Dr.-Ing. Wolfgang Lehner** Technische Universität Dresden Fakultät Informatik Institut für Systemarchitektur Lehrstuhl für Datenbanken 01062 Dresden
	Prof. Themis Palpanas University of Paris LIPADE 45, rue des Saints-Pères 75006 Paris Frankreich
Tag der Verteidigung:	9. März 2020

ABSTRACT

Time series analytics is a fundamental prerequisite for decision-making as well as automation and occurs in several applications such as energy load control, weather research, and consumer behavior analysis. It encompasses time series engineering, i.e., the representation of time series exhibiting important characteristics, and data mining, i.e., the application of the representation to a specific task. Due to the exhaustive data gathering, which results from the "Industry 4.0" vision and its shift towards automation and digitalization, time series analytics is undergoing a revolution. Big datasets with very long time series are gathered, which is challenging for engineering techniques. Traditionally, one focus has been on raw-data-based or shape-based engineering. They assess the time series' similarity in shape, which is only suitable for short time series. Another focus has been on model-based engineering. It assesses the time series' similarity in structure, which is suitable for long time series but requires larger models or a time-consuming modeling. Feature-based engineering tackles these challenges by efficiently representing time series and comparing their similarity in structure. However, current feature-based techniques are unsatisfactory as they are designed for specific data-mining tasks.

In this work, we introduce a novel feature-based engineering technique. It efficiently provides a short representation of time series, focusing on their structural similarity. Based on a design rationale, we derive important time series characteristics such as the long-term and cyclically repeated characteristics as well as distribution and correlation characteristics. Moreover, we define a feature-based distance measure for their comparison. Both the representation technique and the distance measure provide desirable properties regarding storage and runtime.

Subsequently, we introduce techniques based on our feature-based engineering and apply them to important data-mining tasks such as time series generation, time series matching, time series classification, and time series clustering. First, our feature-based generation technique outperforms state-of-the-art techniques regarding the accuracy of evolved datasets. Second, with our features, a matching method retrieves a match for a time series query much faster than with current representations. Third, our features provide discriminative characteristics to classify datasets as accurately as state-of-the-art techniques, but orders of magnitude faster. Finally, our features recommend an appropriate clustering of time series which is crucial for subsequent data-mining tasks. All these techniques are assessed on datasets from the energy, weather, and economic domains, and thus, demonstrate the applicability to real-world use cases. The findings demonstrate the versatility of our feature-based engineering and suggest several courses of action in order to design and improve analytical systems for the paradigm shift of Industry 4.0.

CONTENTS

1 INTRODUCTION **11**

2 FOUNDATIONS OF TIME SERIES ANALYTICS **15**

2.1 Time Series and Domains 15
2.1.1 Time Series in Energy 16
2.1.2 Time Series in Meteorology and Climate 17
2.1.3 Time Series in Medicine 18

2.2 Data-mining Tasks 18
2.2.1 Time Series Generation 19
2.2.2 Time Series Matching 19
2.2.3 Time Series Classification 19
2.2.4 Time Series Clustering 20

2.3 Challenges 20

2.4 Summary 21

3 TIME SERIES ENGINEERING **23**

3.1 Raw-data-based Engineering 24

3.2 Shape-based Engineering 26
3.2.1 Representation 26
3.2.2 Distance 28

3.3 Model-based Engineering 29
3.3.1 Representation 30
3.3.2 Distance 35

3.4 Feature-based Engineering 36
3.4.1 Representation 36
3.4.2 Distance 39

3.5 Summary 39

4 FEATURE-BASED ENGINEERING ACROSS DATA-MINING TASKS **41**

4.1 Design Rationale 42

4.2 Time Series Model 43

4.3 Decomposition 43

4.3.1 Related Work . . . 44
4.3.2 Multi-seasonal Decomposition . . . 44
4.4 Feature-based Representation . . . 45
4.4.1 Features for Deterministic Components . . . 46
4.4.2 Features for Stochastic Component . . . 47
4.4.3 Representation Size . . . 47
4.4.4 Representation Time . . . 48
4.5 Feature-based Distance Measure . . . 49
4.6 Summary . . . 50
5 TIME SERIES GENERATION 51
5.1 State of the Art . . . 52
5.1.1 Properties of Generation Techniques . . . 52
5.1.2 Raw-data-based Generation Techniques . . . 53
5.1.3 Model-based Generation Techniques . . . 54
5.1.4 Assessing Expressiveness . . . 56
5.1.5 Comparison . . . 57
5.2 Feature-based Generation . . . 58
5.2.1 Feature-based Modification . . . 58
5.2.2 Feature-based Recombination . . . 60
5.2.3 Comparison . . . 64
5.3 Experimental Evaluation . . . 64
5.3.1 Experimental Setting . . . 65
5.3.2 Feature-based Distance . . . 66
5.3.3 Standard Distance . . . 68
5.4 Summary . . . 71
6 TIME SERIES MATCHING 73
6.1 State of the Art . . . 74
6.1.1 Original SAX . . . 75
6.1.2 SAX Extensions . . . 76
6.2 Season- and Trend-aware Symbolic Approximation . . . 77
6.2.1 Season-aware Symbolic Approximation . . . 78
6.2.2 Trend-aware Symbolic Approximation . . . 81
6.2.3 Properties of Engineering Techniques . . . 83
6.3 Experimental Evaluation . . . 83
6.3.1 Experimental Setting . . . 84
6.3.2 Results and Discussion . . . 87
6.4 Summary . . . 91
7 TIME SERIES CLASSIFICATION 93

7.1 State of the Art . . . 94
7.1.1 Run Length Distribution . . . 95
7.1.2 Discrete Wavelet Transform . . . 95
7.1.3 Large Feature Vector . . . 96
7.2 System Overview . . . 96
7.2.1 Labeled Dataset . . . 97
7.2.2 Feature-based Representation . . . 97
7.2.3 Normalization . . . 98
7.2.4 Feature Selection . . . 99
7.2.5 Feature-based Classification . . . 99
7.3 Experimental Evaluation . . . 100
7.3.1 Experimental Setting . . . 100
7.3.2 Results and Discussion . . . 103
7.4 Summary . . . 108

8 TIME SERIES CLUSTERING 109

8.1 Cross-sectional Autoregression Model . . . 110
8.1.1 Integration . . . 110
8.1.2 Autoregression . . . 111
8.1.3 Error Terms . . . 111
8.2 Feature-based Clustering . . . 112
8.2.1 ACF and PACF for ARIMA . . . 112
8.2.2 ACF and PACF for CSAR . . . 112
8.3 Experimental Evaluation . . . 113
8.3.1 Experimental Setting . . . 113
8.3.2 Results and Discussion . . . 116
8.4 Summary . . . 118

9 CONCLUSIONS 119

BIBLIOGRAPHY 123

LIST OF FIGURES 133

LIST OF TABLES 135

A PROOFS FOR sSAX AND tSAX 137

A.1 Proof of Lower-bounding sPAA . . . 137
A.2 Proof of Lower-bounding sSAX . . . 139
A.3 Proof of Combined Trend Feature . . . 140
A.4 Proof of Lower-bounding tPAA . . . 140
A.5 Proof of Lower-bounding tSAX . . . 141

B LIST OF SYMBOLS 143

ACKNOWLEDGMENTS

First and foremost, I would like to thank Wolfgang Lehner for giving me the opportunity to realize this thesis project. As my advisor, he guided my research project, provided many ideas as well as valuable feedback. He also gave me the time and freedom I needed to evolve my thesis plan. Because of him, I was able to attend exciting conferences and workshops and to gain experience with industrial partners. Thanks for everything!

I want to thank Themis Palpanas for co-refereeing this thesis. Moreover, I am deeply grateful to my colleagues Claudio Hartmann and Martin Hahmann, who acted as co-advisors over the last years. Claudio took a lot of time for discussions concerning my research, as he is also deeply interested in time series analytics. He was also a great roommate and created a productive working atmosphere. I am also thankful for our co-operation in time series forecasting and clustering. Martin offered me a lot of support in publishing papers, especially in finding an appealing and convincing writing style. With his great sense of humor, he enriched every conversation, and I am looking forward to his conference on surreal computer sciences. I also want to thank my former supervisors, who inspired me with their work on time series analytics while I was studying: Philipp Rösch and Lars Dannecker from SAP, and especially Ulrike Fischer.

This thesis would not have been possible without the support from the team. I am thankful to all my colleagues for a great and creative atmosphere that included constructive discussions as well as fun coffee breaks. Thank you, Alex, Annett, Axel, Dirk, Elvis, Johannes P., Johannes L., Julius, Lisa, Maik, and Patrick! Special thanks also to Robert and Lucas for many fruitful collaborations on student courses and theses. I am deeply grateful to Ioana Manolescu, who provided me "academic shelter" for one year; her research group, especially Alexandre, Félix, Khaled, Maxime, Mikaël, Mirjana, Paweł, and Tayeb, received me well, and I enjoyed the social activities. A special thanks to all students who contributed to my research projects. Moreover, I would like to thank Claudio, Jiři, and Olga for proof-reading this thesis and for providing many valuable comments.

Finally, I am deeply grateful for the constant encouragement from my family and friends. My parents Ilona and Lutz, as well as my sister Anita always stood behind me and supported me during this tough time. Moreover, I enjoyed unforgettable activities with my friends and band. The music we played did not only made our audience happy; it also made me happy. Thank you for helping me keep a work-life balance!

Lars Kegel
Dresden, January 9, 2020

1 INTRODUCTION

EXHAUSTIVE DATA GATHERING is not a new trend anymore but can be considered standard practice in many domains. It is the primary driver of the current paradigm shift in industrial production towards automation and digitalization. Especially the German-speaking area refers to this shift as *Industry 4.0* [LFK+14]. For example, *smart factories* follow this paradigm shift; they are heavily equipped with sensors providing information to autonomous factory systems. Besides, parameters of *cyber-physical systems*, which merge physical and digital components, are monitored for controlling and maintenance. Finally, systems connected via the *Internet of Things* (IoT) monitor and communicate their status [XYWV12].

A significant part of these measured values is captured over time and thus forms a time series [SS11]. A dataset of these time series gives analytical insights into the underlying processes, which are uncovered by data mining. Four insights are fundamental for this work: the extraction of important time series characteristics and their reproduction [MS82], the retrieval of a similar time series [AFS93], the mapping of a time series to a class label [PO94], and the partitioning of a dataset into meaningful clusters [Bel77]. We refer to these four data-mining tasks as time series generation, time series matching, time series classification, and time series clustering, respectively. In recent years there has been growing interest in carrying out these data-mining tasks in many domains [KHL18, ZP18, BLB+17, ASY15].

It is challenging to carry out these data-mining tasks for two reasons. First, time series are inherently high-dimensional. Their discriminative characteristics do not arise from one single value, but from many, possibly very distant values. Moreover, these characteristics do not only appear by considering each value in isolation, but also from the mutual dependence of values. Since data-mining tasks usually focus on low-dimensional data types, they are not directly applicable to time series and their specific nature. Second, exhaustive data gathering leads to big datasets. Not only do time series occur together with thousands of other time series, but they also have a fine granularity, leading to large series with tens of thousands of values. Thus, they require much storage, and their comparison is time-consuming. Overall, techniques are required that transform time series in a low-dimensional representation, while enabling effective and efficient data mining.

In the literature, the term *engineering* refers to the branch of science and technology that focuses on the design, the building, and the use of structures [Lex19]. With this in mind, we introduce the term *time series engineering* to refer to the task of designing a *representation technique* that builds a low-dimensional *representation* of a time series and of designing a *distance measure* which describes how far away two representations are by returning

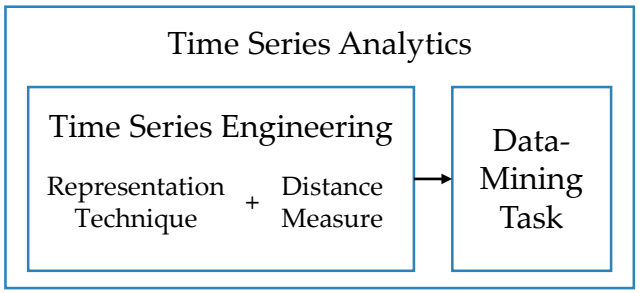


Figure 1.1: Time Series Analytics

their *distance* (Figure 1.1). Thus, time series engineering provides techniques to transform a time series into a low-dimensional representation, exhibiting its important characteristics. Consequently, the term *time series analytics* encompasses time series engineering and the application to a data-mining task.

The energy, meteorological, and economic domain are attracting considerable interest due to the rise of cyber-physical systems and IoT, and we focus on them in our research projects [FFQ17, GOF17]. In the energy domain, a multitude of smart meters and energy management systems captures processes as time series, such as the electricity consumption in households and businesses. This data is gathered by the grid operator and provides analytical insights to control the load in the grid [UFLD13]. Meteorological time series are heavily analyzed to assess the feasibility of industrial installations, such as wind or solar power plants [MS82]. Besides, climate phenomena are also captured as time series. For example, they provide insights about dependent ecosystems [STK+03]. In the economic domain, macro-economic, sales, and payment data is gathered as time series and analyzed for better decision-making [MSA18]. For example, many shops monitor payment transactions to analyze and forecast consumer behavior [Int17]. Since these domains also face the challenges of big datasets it is of highest importance to apply a suitable engineering technique to capture the important characteristics of these datasets.

There is a considerable amount of literature on engineering techniques. While raw-data-based techniques focus on the time series as is, applying different measures to describe their distance, shape-based and model-based techniques reduce a time series to a low-dimensional space, by selecting important points, aggregating segments of the time series, or estimating a generative model.

Feature-based techniques form the fourth class of engineering techniques. A feature is a global time series property, which arises from the application of a method from the time-series analysis literature [FLJ13]. These features are gathered as a feature vector, characterizing a time series by its important characteristics. This class of engineering techniques is very promising for big datasets, as it focuses on global structural properties and calculates them efficiently. As such, different feature-based techniques have been applied to various data-mining tasks, such as generation, matching, classification, and clustering. However, they cannot be easily adopted by other data-mining tasks, as they do not support their specific requirements. A generation technique must evolve new time series from a representation, while a classification technique requires discriminative features for accurately mapping time series to their correct class label. Time series matching requires a distance measure for an efficient retrieval, while clustering relies on a distance measure for grouping similar time series and separating dissimilar ones. These requirements have not yet been addressed altogether.

With this in mind, our goal is the design of a feature-based engineering technique that is used across data-mining tasks. First, it has to capture important characteristics of time

series. Second, it should provide desirable properties regarding space and runtime, i.e., it should efficiently build a short representation and compare them with an efficient distance measure, and thus, tackle the challenges of big datasets. Finally, it should consider specific requirements from data-mining tasks and provide analytical insights effectively and efficiently.

Summary of Contributions

Overall, we propose a feature-based engineering technique that efficiently captures the long-term and cyclically repeated characteristics of a time series in a short representation, together with its distribution and correlation characteristics. Its distance measure focuses on the global structural similarity, which is desirable for big datasets with long time series [WSH06]. Moreover, we show its versatility to handle fundamental data-mining tasks. In more detail, the contributions of this work are:

- We give an overview of domains where time series analytics plays an important role, and exemplify data-mining tasks on three selected domains. Moreover, we motivate time series engineering by formulating the challenges of data-mining tasks regarding big time series datasets.
- We survey engineering techniques from the literature and review them regarding desirable properties for the application on big time series datasets. We conclude that a versatile feature-based technique is required to tackle a variety of data-mining tasks.
- We introduce a feature-based engineering technique, which applies to a multitude of domains and across data-mining tasks. Moreover, we demonstrate its competitive properties compared to other techniques [KHL17b, KHL18].
- We give an overview of time series generation by surveying generation techniques and comparing the characteristics they are able to reproduce. Moreover, we propose feature-based generation techniques that reproduce these characteristics accurately [KHL16, KHL17a, KHL18].
- We extend the state-of-the-art engineering technique for time series matching, the symbolic aggregate approximation [LKLC03], with features from our engineering technique, which leads to a significantly more effective and efficient matching without increasing the representation size.
- Regarding time series classification, we survey engineering techniques for the classification of long time series. Taking into account both effectiveness and efficiency, we propose a feature-based classification that classifies as accurately as state-of-the-art techniques, but orders of magnitude faster.
- We assess a feature-based clustering of a dataset, applied to time series forecasting. In particular, we show that a vectorized forecast technique [Har18], estimating one model for a cluster of time series, is more accurate if the time series in a cluster have similar features [HKL20].

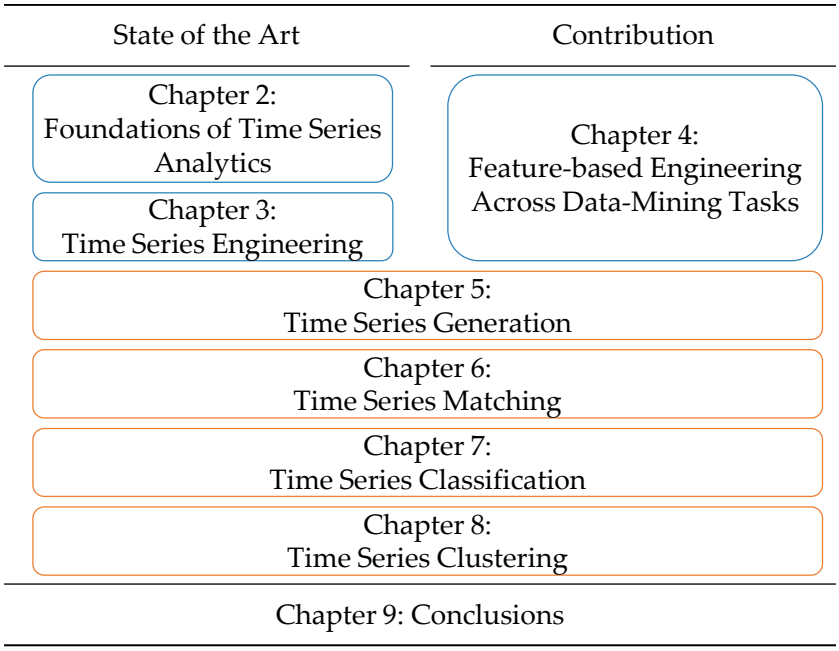


Figure 1.2: Structure of this Work

Structure of this Work

Figure 1.2 illustrates the overall structure of this work. In the first part, we give the necessary background on time series engineering. Chapter 2 starts with the foundations of time series analytics by giving example applications of time series and by explaining the challenges of data mining. Chapter 3 surveys engineering techniques for time series. Based on these observations, we introduce our engineering technique in Chapter 4 along with a feature-based representation technique and distance measure. In the second part, we shift our attention to four selected data-mining tasks, i.e., time series generation (Chapter 5), time series matching (Chapter 6), time series classification (Chapter 7), and time series clustering (Chapter 8). We start each of these chapters by introducing state-of-the-art engineering techniques and desirable properties. Subsequently, we apply our engineering technique to the data-mining task and evaluate it experimentally. We finally conclude this work in Chapter 9 with a summary and challenges for future research activities.

2

FOUNDATIONS OF TIME SERIES ANALYTICS

THIS CHAPTER gives an overview of time series analytics. It introduces time series as the fundamental data type for this work and identifies domains that gather this data for different purposes (Section 2.1). Subsequently, it presents data-mining tasks for time series and exemplifies the relevance of four data-mining tasks, time series generation, matching, classification, and clustering in several domains (Section 2.2). Finally, it identifies challenges of big time series datasets, which lead researchers to carry out time series engineering (Section 2.3). We conclude with a summary of our observations (Section 2.4).

2.1 TIME SERIES AND DOMAINS

It is quite natural in many domains to measure a process at consecutive time instances. The measured values are likely to depend on each other instead of being independent and identically distributed [SS11]. The data type that stores measured values is a *time series*. It covers the *value domain* by storing each value as a real number, and it covers the *time domain* by storing the values in increasing order of time. Formally, a time series $\underline{y}$ is a vector of values y that are measured at discrete time instances t:

$$\underline{y}^\intercal = (y_1, \ldots, y_t, \ldots, y_T) \text{ where } \underline{y} \in \mathbb{R}^T, t \in \mathbb{N}_{>0}, t \leq T \tag{2.1}$$

The distance between two time instances is called *granularity*. In this work, we assume that a time series (1) is finite with a fixed length T, (2) is complete, i.e., there are no missing values, and (3) is equidistant, i.e., the distance between between any two consecutive time instances is constant. Works that omit these conditions, i.e., omit condition (1) and process time series in a streaming fashion or omit conditions (2) and (3) and process irregular time series, are orthogonal and not covered in this work. Conditions (2) and (3) are achieved by cleaning and transforming the data beforehand.

Often, processes that are related to each other are bundled together. Consequently, they are stored as a *time series dataset* which is a set of I time series:

$$Y = \{\underline{y}_1, \ldots, \underline{y}_i, \ldots, \underline{y}_I\} \text{ where } i \in \mathbb{N}_{>0}, i \leq I \tag{2.2}$$

where we assume that all time series of a dataset have the same length.

There is a multitude of domains that use time series for data mining [SS11, Fis14]. As shown in Figure 2.1, they may be broadly categorized into four groups: *Economy*, *Inanimate Nature and Environment*, *Biometrics*, and *Society*.

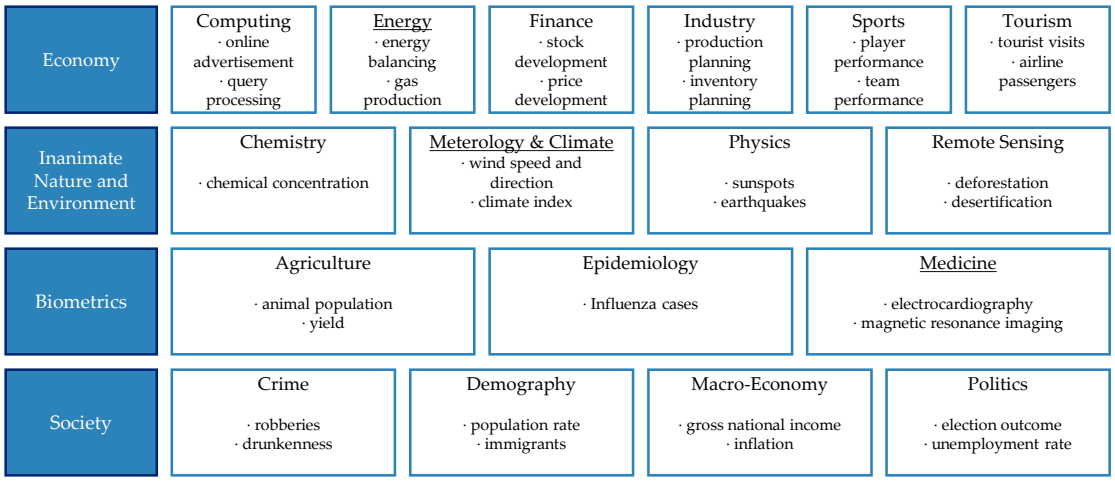


Figure 2.1: Domains of Time Series

The first group, Economy, encompasses domains such as computing, energy, finance, industry, sports, and tourism. These domains utilize time series for market research and for supporting business processes. The energy domain is attracting widespread interest due to the increased installations of smart metering and energy management systems that measure many electric processes at a fine granularity for energy load control. Therefore, we give a brief overview of two data-mining tasks in this domain (Subsection 2.1.1).

In the second group, Inanimate Nature and Environment, researchers study the natural sciences (chemistry, physics), and environmental sciences (meteorology and climate, remote sensing). In these domains, time series express natural and environmental phenomena over time and allow decision-makers to take actions such as planning new industrial sites or adopting new environmental policies. In meteorology and climate, big time series datasets that contain weather and climate influences are most interesting for an analysis, which is why we give an overview of data-mining tasks in this domain (Subsection 2.1.2).

Biometrics is the application of statistical analysis to biological data. It encompasses domains such as agriculture, epidemiology, and medicine. In medicine, time series support medical diagnosis and data-mining tasks are among the most commonly discussed in the literature, which is why we present three data-mining tasks from this domain (Subsection 2.1.3).

The last group, Society, studies the social sciences and social relationships. It encompasses domains such as crime, demography, macro-economy, and politics. For example, demography utilizes time series for grouping societies with similar development of their population. These insights are used by decision-makers for adopting new policies.

Subsequently, we present scenarios from three selected domains, energy, meteorology and climate, as well as medicine, and demonstrate the use of time series for their purposes.

2.1.1 Time Series in Energy

By 2020, the European Union aims at producing at least 20% of its total energy using renewable energy sources (RES) [Eur09], and by 2030, it has even higher ambitions [Eur19]. The impact of RES on the electricity sector is the highest compared to other sectors

[Eur19]. However, the feed-in of RES is challenging due to their intermittent and irregular nature. Solar power is produced only during the day and not during the night while, in Germany, wind energy is produced mostly in winter. The energy production is more and more decentralized, but the existing grid was initially designed for unidirectional power flows, i.e., from power plants to consumers.

To tackle these challenges, several technologies have been proposed that are innovated and rolled out piece by piece. Among them, smart meters and data management platforms contribute to a balanced energy grid, and they do this by time series analytics. Smart meters record and transmit the energy consumption of a household, a circuit, or a device to improve energy savings. One approach is to notify a consumer about the correct, inefficient, or even faulty behavior of a device and recommend him to take action. Thus, the monitored device is the process whose measured values are captured as a time series. A classification model is trained to identify a device and its behavior automatically [LBCSA11]. Grid authorities may also store time series from smart meters on a data management platform. On this aggregated level, they use time series to provide better analytical insights and to balance the energy grid [UFLD13]. Moreover, they have to take care that the data management platform fulfills the specific requirements of their area, i.e., the correct sizing and performance. Therefore, Arlitt et al. propose an assessment tool that involves time series generation. This tool helps grid authorities to assess their data management platform by loading and analyzing generated datasets with realistic characteristics, but configurable in their size [AMB+15].

Thus, we identify time series classification and generation as crucial data-mining tasks in the energy domain.

2.1.2 Time Series in Meteorology and Climate

Meteorological time series capture a variety of atmospherical phenomena such as temperature, solar irradiation, wind speed, wind direction, and precipitation. They are most frequently used for estimating weather prediction models that provide forecasts of these phenomena. Besides forecasting, these time series are also used in two other data-mining tasks, which are time series generation and time series clustering.

Systems that rely on atmospherical phenomena have to be assessed in order to test and verify their performance, their robustness, and their correct sizing. For example, wind power plants are heavily evaluated for possible weather scenarios (wind speed and wind direction at different altitudes) before they are installed at a specific location. If such scenario datasets are not available because measuring in the field is expensive or not possible, time series with realistic characteristics are generated [MS82].

An essential task in climate research is the discovery of climate phenomena linked to each other. These so-called teleconnections give explanations on how climate changes and how ecosystems respond to remote phenomena. Usually, they are discovered by climate indices capturing the variance on a regional and global level as a time series. It has been shown that clustering climate indices are an essential tool of discovering teleconnections [STK+03].

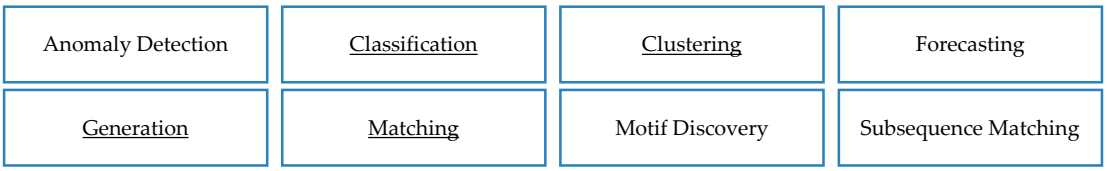


Figure 2.2: Data-mining Tasks

2.1.3 Time Series in Medicine

In medicine, diagnostic tools capture a variety of characteristics from the human body. Two prominent examples are the electrocardiography (ECG) of the heart and the sequential analysis of the deoxyribonucleic acid (DNA).

The ECG measures the heartbeat of a patient by its electrical activity. The values represent time series that allow physicians to identify heart arrhythmia, i.e., conditions that result from disturbances of the heartbeat. Since some arrhythmias appear infrequently, patients have to be monitored over several days. Subsequently, physicians analyze the resulting time series, which can be very time-consuming. Therefore, research focuses on the automatic classification to support this task [DOR04].

Another diagnostic tool for medical but also for biological research is the sequential analysis of the DNA, i.e., the determination of the nucleotides adenine, guanine, cytosine, and thymine in their sequential order. It enables researchers to discover homologies between species such as humans and monkeys. Since the DNA is very long and split into several chromosomes, it is a challenging task to match subsequences to each other in order to discover these homologies. Camerra et al. propose an approach where they translate a DNA sequence into real numbers forming a time series [CPSK10]. They build a time series index that stores representations of all subsequences of a monkey DNA. Then, they query the index with subsequences from human DNA and find matches. These matches establish a co-occurrence map identifying the chromosomes of humans and monkeys that are most similar. Not only do their results agree with previous research, their time series index is also an efficient matching method.

Thus, time series classification and matching are crucial data-mining tasks in medicine.

2.2 DATA-MINING TASKS

Time series have been used in several data-mining tasks, as presented in Figure 2.2. *Anomaly detection*, *clustering*, *generation*, *matching*, *motif discovery*, and *subsequence matching* are descriptive tasks, which analyze past and current data in order to prepare informed decisions. *Classification* and *forecasting* are predictive tasks; they infer the class of an unlabeled time series and the future values of a given time series, respectively. Based on the examples mentioned above and their requirements, we focus on four selected data-mining tasks, explain and motivate them in the following subsections.

2.2.1 Time Series Generation

Time series generation extracts important characteristics from a given dataset and reproduces them in a generated dataset. While this looks like a paradox considering the abundance of data that is collected, it is still a very important task for evaluating a system and for providing evolved data in case there is no given data available.

Thus, it has been applied in a multitude of domains. As reported earlier, it is used in the meteorological domain for generating wind speed time series [MS82]. Since then, researchers have been developing techniques for simulating further weather parameters [JL86, KKD91, BdMK02, MH15]. The energy domain utilizes it to assess renewable energy power plants [JL86, ILD+17], while industry and computing apply generated datasets for various evaluation purposes [CDB94, SJ13].

2.2.2 Time Series Matching

Time series matching retrieves a time series from a dataset that is most similar to a query time series [KK03]. A naive matching algorithm compares the query time series to each time series from the dataset one by one. However, this approach is time-consuming which is why matching methods focus on a more efficient retrieval. Efficiency can be increased by pruning unpromising observations as early as possible [AFS93].

The data-mining task was first mentioned in 1993 where Agrawal et al. presented an R*-index that stores coefficients of the Fourier transform of time series [AFS93]. It gained popularity by the GEMINI approach (GEneric Multimedia INdexIng) where researchers built upon their work on time series engineering [Fal96, KK03]. Since then, progress has been made towards an effective and efficient time series engineering [LKLC03], i.e., a better pruning and a faster distance calculation, and towards more efficient indexing structures [SK08]. Up to now, index structures are improved regarding build time and contiguity [CPSK10, KDZP18].

Time series matching is widely applied in domains such as industry, finance [AFS93], and medicine. The matching of DNA sequences is a prominent example [CPSK10].

2.2.3 Time Series Classification

Time series classification is a supervised data-mining task that maps an unlabeled time series to a class label [KK03]. It is applied for two reasons: to support human classification decision [DOR04] and to fully automatize processes by, for example, triggering events in case of irregular time series [PO94].

Usually, classification is carried out for observations from a low-dimensional space. However, time series are high-dimensional and their values are mutually dependent [KK03].

Due to these differences, time series classification may considered a proper data-mining task. As such, it was first discussed in an automated labeling of control charts in 1994 [PO94]. It gained more attention as researchers started proposing highly discriminative distance measures that classified their datasets well [KK03]. Up to 2017, researchers also focused on better time series engineering and on combining classification techniques [BLB+17].

Following Chen et al. [CKH+15], classification problems mainly arise from computing where signals from images, sensors, and motions are observed, from medicine where the automated labeling of the aforementioned ECG signals but also of, e.g., blood flow dynamics support the diagnosis, and from energy where electrical devices are classified by their signal in order to detect failures.

2.2.4 Time Series Clustering

Time series clustering is an unsupervised data-mining task. Its goal is to partition a time series dataset into clusters such that each cluster is homogeneous, i.e., the distance of time series within a cluster is minimal, while the distance of time series from different clusters is maximal [Lia05]. As pointed out in [ASY15], there are several reasons to carry out this data-mining task. (1) Clusters may uncover interesting patterns in a dataset, thus they help to identify frequent patterns that arise commonly as well as suprising patterns that arise rarely. (2) Clustering supports the exploration of big time series datasets by partitioning big datasets and presenting only a prototype time series per cluster. Moreover, these clusters s may be hierachically ordered such that they may be collapsed and uncollapsed. (3) It is applied as a subroutine in other data-mining tasks such as time series forecasting. (4) The visualization of cluster structures helps to quickly understand the structure of the data along with its regular and irregular behavior.

Due to the special structure of time series, time series clustering is considered a proper data-mining task. Most often, clustering techniques are applied in a low dimension, too. However, time series engineering is necessary in order to prepare a dataset such that it presents the important characteristics to the clustering technique [Lia05].

Time series clustering was first mentioned in a work in 1977 [Bel77]. Since then, the data-mining task gained more and more attention because different techniques of time series engineering were proposed and different clustering algorithms were applied [Lia05, ASY15]. Moreover, it is utilized in a multitude of domains. Among them, it has been applied in the climate domain to discover the aforementioned climate indices [STK+03], in the energy domain to discover similar energy consumption behavior [KB90], and in medicine to partition ECG data [PG15].

2.3 CHALLENGES

Data-mining tasks for time series have to tackle two major challenges. First, they have to take many values into account that are ordered by time because a time series is an inherently *high dimensional* data type. Second, the paradigm shift of Industry 4.0 leads to an increasing number of sensors, and these processes are captured at a finer granularity. Thus, data-mining tasks face *big time series datasets*. Subsequently, these two challenges are further detailed.

High Dimensionality

It is easier to carry out data-mining tasks once on a whole time series rather than once per time instance. The latter case would require to study the evolution of data-mining results across the time instances [Bel77]. Therefore, most publications focus on the former case and present whole time series to a data-mining technique, i.e., they present high-dimensional data.

One challenge arising from this presentation is the *ordering* of the values by time. The important characteristics of a time series do not only arise by considering each value in isolation but also from the mutual dependence of values. Distance measures have to describe this dependence accurately [KK03]. Besides the ordering, the *length* of a time series is a second challenge. Standard distance measures used for low-dimensional data cannot be effectively applied to long time series because the noise in the time series disturbs their results [KK03], and their notion of similarity becomes dubious [WSH06].

Data-mining tasks that take time series as is and that rely on standard distance measures become intractable: (1) Generation techniques aim at evolving time series that are similar to given time series regarding their shape, their value distribution, and their correlation [BdMK02]. Standard distance measures do not describe distribution and correlation characteristics and thus fail to provide this similarity. (2) Indexes for matching time series cannot handle more than 16-20 dimensions, i.e., time series values. Time series are usually much longer and the index would degenerate [LKLC03]. (3) Time series classifiers could be prepared to work in a high-dimensional space. However, authors suggest reducing a time series rather than constructing complex classifiers [BDHL12]. (4) Some clustering techniques cannot handle high-dimensional data and would not accomplish their task on time series datasets [WSH06]. It is thus beneficial for these data-mining tasks to reduce the dimensionality of time series.

Big Time Series Datasets

Besides the high dimensionality, time series datasets are challenging due to their size. Once datasets are too big to fit into memory, data-mining tasks decrease in speed due to much disk I/O [ASY15]. Therefore, time series matching methods represent time series in a compressed manner so that they fit into memory. Due to pruning strategies, disk I/O is further reduced [LKLC03]. Other data-mining techniques such as highly accurate classifiers fail due to their complexity [BLHB15]. Clearly, they are not feasible for time series with a fine granularity.

2.4 SUMMARY

Although time series occur in many domains and provide insights for different purposes they are challenging due to their inherent nature and due to the increasing dataset size. Thus, data-mining tasks cannot be effectively and efficiently applied to provide analytical insights. These observations suggest that time series should be transformed in order to expose their important characteristics in a small representation.

3

TIME SERIES ENGINEERING

ENGINEERING TECHNIQUES consist of two components: first, a representation technique that builds representations of time series datasets and second, a distance measure that uses representations to compare time series with each other. They tackle the challenges of big time series datasets for the following reasons. A representation technique reduces a time series to a low-dimensional space and thus, a time series dataset has a smaller memory footprint. It provides representations that capture important characteristics. The distance measure ensures that these representations are similar if the original time series are similar [ASY15]. Moreover, the distance calculation is faster on a representation than on the original time series.

Figure 3.1 gives an overview of engineering techniques that are commonly applied in the literature. They are classified into *raw-data-based*, *shape-based*, *model-based*, and *feature-based* approaches. While raw-data-based and shape-based engineering focuses on a *similarity in shape*, i.e., on a similarity of time-dependent characteristics, model-based and feature-based engineering focuses on a *similarity in structure*, i.e., on a similarity of time-independent characteristics. Beyond this classification, we characterize the engineering techniques regarding five properties that express how they represent and compare time series. It is desired that a technique fulfills all of these properties in order to be applicable for big time series datasets and for a variety of data-mining tasks.

Representation Size The representation size of a time series should be small compared to its original size. Big time series datasets may not fit into memory and carrying out a data-mining task may incur additional disk I/O. We assess this property by the storage size in bit that one representation is occupying. We assume that a real value is stored as a floating-point value occupying 32 *bit*.

Representation Time A representation technique should provide a fast representation of a time series. We assess this property by counting the number of passes the technique has to read a time series dataset.

Distance Storage Usually, the distance is calculated directly on the representations. But some distance measures require additional storage for the calculation. We assess this property by the size in byte of the storage overhead which is required once per time series dataset.

Distance Time Data-mining tasks can benefit from a representation technique if the comparison between representations is fast. We assess this property by counting the value comparisons between two representations.

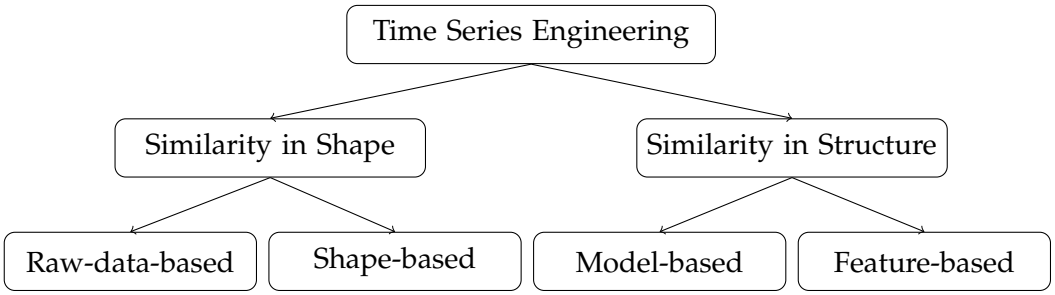


Figure 3.1: Taxonomy of Engineering Techniques

Lower-bounding Distance Measure A distance measure is lower-bounding if the distance of two representations is always smaller than or equal to the *Euclidean distance* of the original time series:

$$d_{ED}(\underline{y}, \underline{y}') = \sqrt{\sum_{t=1}^{T} (y_t - y'_t)^2} \tag{3.1}$$

This property is a desirable property in time series matching where observations can be efficiently pruned: if its distance to a query is too large then there is no need to evaluate their Euclidean distance. It also implies that the Euclidean distance is always considered the *true* distance between time series. Agrawal et al. point out that this true distance depends on the data-mining task and the domain [AFS93]. However, they claim several advantages of the Euclidean distance over other distance measures, most importantly, it preserves the distance if the time series are transformed by rotation, translation, or reflection. There are also drawbacks of the Euclidean distance: first, it focuses on a similarity in shape and second, it is sensitive to noise and misalignments. When designing a lower-bounding distance measure we accept these drawbacks in order to compare our solution to the literature.

In the following sections, we present and review engineering techniques (Sections 3.1-3.4) and conclude with an overall comparison of their properties (Section 3.5).

3.1 RAW-DATA-BASED ENGINEERING

Raw-data-based engineering takes a time series as is and does not represent it in a low-dimensional space. Instead, it focuses on distance measures, which can be distinguished into *lock-step*, *elastic*, and *cross-correlation distance measures* [Lia05, DTS+08]. Further distance measures are often combinations of these categories [ASY15].

Lock-step Distance Measure

Lock-step distance measures compare the t-th value of one time series with the t-th value of another time series [DTS+08]. The *Minkowski distance*, defined as

$$d_{Mink}(\underline{y}, \underline{y}') = \left(\sum_{t=1}^{T} |y_t - y'_t|^o \right)^{1/o} \tag{3.2}$$

gathers several lock-step distance measures. For $o = 2$, it is equivalent to the Euclidean distance $d_{ED}(\underline{y}, \underline{y}')$ which is its most prominent example. It is applied in time series

clustering [Lia05, ASY15], in time series matching [DTS+08] and in time series classification [DTS+08, BLB+17]. The *root mean squared error* (RMSE) is derived from the Euclidean distance and is often used in time series generation for comparing generated and given time series [BdMK02, BK09, APHRH13, SJ13]:

$$d_{RMSE}(\underline{y}, \underline{y}') = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - y'_t)^2} \tag{3.3}$$

Other Minkowski distances such as the *Manhattan distance* d_{MD} ($o = 1$) and the *Chebyshev distance* ($o = \infty$) are applied in time series matching [DTS+08] but they are less common. Overall, lock-step distance measures have a linear complexity and they requires T comparisons for the distance calculation without any distance storage. They are easy to implement, and they are parameter-free which is why they are often a baseline for carrying out data-mining tasks or for assessing their results [CKH+15, KHL18]. The Euclidean distance is lower-bounding itself [LKLC03], other distance measures do not provide this property.

Elastic Distance Measure

The fixed mapping of lock-step distance measures makes them sensitive to noise and misalignments. Elastic distance measures tackle this problem by aligning time series with different local speeds and thus, compares one-to-many or one-to-none values [DTS+08]. They are further distinguished into *dynamic time warping* and *edit distance measures*.

Dynamic Time Warping Dynamic time warping (DTW) aligns two time series such that their distance is minimized [SC78]. It uses a matrix of size $T \times T$ which has the lock-step distance of values y_t and $y'_{t'}$ in cell (t, t'). Within this matrix, DTW searches a warping path $WP = \{d_1, d_2, ..., d_k, ..., d_K\}$ that consists of the distances of matrix cells. For restricting the search space, the warping path has to fulfill three conditions [SC78]: first, it starts and ends in the corners $(1, 1)$ and (T, T), respectively, second, it is continuous, i.e., the path steps from one cell to an adjacent cell, and third, it is monotonous, i.e., the cells are monotonically ordered with respect to the time instances. Constrained DTW (cDTW) further restricts the warping path to a band, i.e., a range of matrix cells. Finally, the DTW distance is:

$$d_{DTW}(\underline{y}, \underline{y}') = min_{WP}\sum_{k=1}^{K} d_k \tag{3.4}$$

Edit Distance Edit distance measures are the second group of elastic distance measures. They are inspired by the edit distance which determines the similarity of two text sequences by taking insertions, deletions, and substitutions into account. The *longest common subsequence* (LCSS) is a variant for time series data [VGK02]. Similar to DTW, it matches time series with different speed but it allows to leave some values unmatched if it would increase the distance. Thus, it claims to be more robust to noise than the Euclidean distance and DTW.

Overall, elastic distance measures are applied in time series matching, classification, and clustering [DTS+08, BLB+17, Lia05, ASY15]. The accurate distance calculation has a quadratic complexity $\mathcal{O}(T^2)$, thus it requires at most T^2 comparisons. However, approaches that use constraint parameters or that calculate approximative distances reduce this complexity down to a linear complexity [VGK02, SC07]. They do not require distance storage and they are not lower-bounding.

Cross-correlation Distance Measure

Cross-correlation distance measures consider time series similar if they are highly correlated, i.e., their linear dependence of each other is very strong. They are based on the Pearson correlation coefficient which is defined as:

$$cc(\underline{y}, \underline{y}') = \frac{\sum_{t=1}^{T}(y_t - m(\underline{y}))(y'_t - m(\underline{y}'))}{\sqrt{\sum_{t=1}^{T}(y_t - m(\underline{y}))^2 \cdot \sum_{t=1}^{T}(y'_t - m(\underline{y}'))^2}} \tag{3.5}$$

where $m(\underline{y}) = {}^1/_T \sum_{t=1}^{T} y_t$ denotes the mean of a time series. The coefficient value ranges between -1 and +1: If $cc = 0$, then there is no linear dependence between the time series. If $cc > 0$, then there is a positive linear dependence. The maximum correlation, $cc = 1$ means that the time series perfectly depend on each other. If $cc < 0$, there is a negative linear dependence between: if one time series increases, the other one decreases. This also leads to perfect anti-correlation, if $cc = -1$.

To use the Pearson correlation coefficient as a distance measure, high correlation values are mapped to 0 and low correlation values are mapped to 1. There are different formulas applied in time series generation and clustering [ILD+17, Lia05, PG15]. Overall, this distance measure needs T comparisons assuming that $m(\underline{y})$ and $m(\underline{y})'$ are known, it does not require further distance storage, and it is not lower-bounding.

3.2 SHAPE-BASED ENGINEERING

Shape-based engineering is not clearly defined in the literature because there is no general definition of a *shape* [PG15]. Aghabozorgi et al. use the term to refer to "working with the raw time-series data" [ASY15], while other authors use it for a reduced representation [AJB97, LKL03]. We follow this second notion and define shape-based engineering as the time-dependent representation of a time series in a low-dimensional space that are considered similar if their shapes as a, e.g., line plot, are similar. Subsequently, we give an overview of *shape-based* representations from the literature along with distance measures and discuss their properties.

3.2.1 Representation

Shape-based representation techniques reduce a time series into a low-dimensional space by *selecting* random or salient values, by *aggregating* values segment-wise, or by *discretizing* values [Fu11].

Random or Salient Points

Sampling is the random selection of values from a time series and the most straightforward technique to reduce the dimensionality in the time domain [Åst69]. It is very efficient as it does not need even one pass over the data. However, it may distort the shape if the sampling is too low [Fu11].

The capture of salient values, i.e., values that are perceptually important leads to a more accurate representation. Intuitively, authors propose the selection of *extreme* values and filter those which are the most extreme among their neighbored values [PF02]. Moreover, they propose the selection of *important* and *critical* values whose filtering is more sophisticated. These representations need at least one pass over the time series [Fu11].

Aggregation

While selection techniques assume that a subset of values represents a time series reasonably well, aggregation techniques assume that all values of a time series should be taken into account but they should be reduced to aggregates. The most prominent representation technique is the *piecewise aggregate approximation* (PAA) [YF00]. It segments a time series into intervals of constant length and aggregates them using their mean value. Let $W \in \mathbb{N}_{>0}$ be the number of segments per time series and W divides the time series length T. The piecewise aggregate approximation $\underline{\bar{y}}$ is the vector of mean values of a time series:

$$\underline{\bar{y}}^\intercal = (\bar{y}_1, ..., \bar{y}_w, ..., \bar{y}_W) \tag{3.6}$$

where

$$\bar{y}_w = \frac{W}{T} \sum_{t=\frac{T}{W}(w-1)+1}^{\frac{T}{W}w} y_t \tag{3.7}$$

The *adaptive piecewise constant approximation* (APCA) also relies on the mean values of intervals but it supports adaptive interval lengths instead of constant ones [KPMP01], requiring many passes over the dataset. However, it has to store the interval length and thus, it can only represent half as many segments as PAA. Moreover, the representation needs many passes over the data while PAA only needs one pass. *Piecewise linear approximation* (PLA) segments a time series into intervals of constant length and represents the values of each segment by linear regression [Keo97]. Like APCA it needs two values to represent each interval but it only needs one pass over the data.

Discretization

The aforementioned shape-based techniques reduce a time series in the *time domain* by representing it with random values, salient values or aggregates. The reduction in the *value domain* is the idea behind discretization. Originally, discretization is carried out on the values themselves, i.e., each value is replaced by a symbol. This is the idea behind the *shape description alphabet* (SDA) which characterizes each value transition with five different states: highly or slightly increasing, stable, and highly or slightly decreasing [AJB97]. Later, such states were utilized to characterize segments instead of values. The *gradient alphabet* characterizes each segment with three states, similarily to SDA [QWW98]. With the aim to flexibly adapt the characterization, the *change ratio* and the *codebook of sequences* (PVQA) introduce an alphabet of size A such that segments are described by A different states [HY99, MLW04]. The final breakthrough is proposed by Lin et al. with the *symbolic aggregate approximation* (SAX) [LKLC03]. SAX combines the PAA aggregation technique with the concept of discretization. Moreover, it assumes that a time series is *z-normalized*, i.e., its values have a mean of zero and a variance of one. Let A be the size of an alphabet ($A \in \mathbb{N}_{>0}$) and let $\underline{b}^\intercal = (b_1, ..., b_a, ..., b_{A-1})$ be a vector of increasingly sorted *breakpoints* that split the real numbers into A intervals:

$$]-\infty, b_1], ...,]b_{a-1}, b_a], ...,]b_{A-1}, \infty[$$

The symbolic aggregate approximation $\underline{\hat{y}}$ is the vector of symbols, i.e., mean values discretized into the alphabet A:

$$\underline{\hat{y}}^\intercal = (\hat{y}_1, ..., \hat{y}_w, ..., \hat{y}_W) \tag{3.8}$$

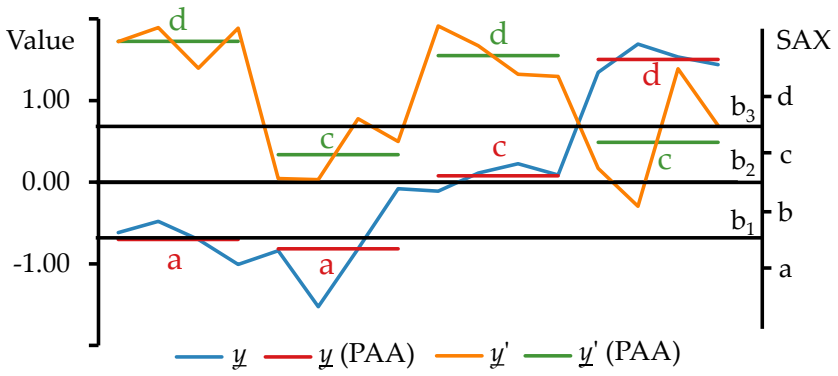


Figure 3.2: Time Series With PAA and SAX Representations

where each mean value is mapped to a discrete value a if its between the corresponding breakpoints:

$$\hat{y}_w = \begin{cases} 1 & -\infty < \bar{y}_w \leq b_1 \\ a & \exists a : b_{a-1} < \bar{y}_w \leq b_a \\ A & b_{A-1} < \bar{y}_w < \infty \end{cases} \tag{3.9}$$

Most importantly, SAX assumes that the PAA mean values of a z-normalized time series is also normally distributed with the same standard deviation. While all presented discretization techniques provide a short representation and only need one pass over the data, only SAX provides versatile and dataset-independent breakpoints.

3.2.2 Distance

Most of the shape-based representation techniques use lock-step distance measures to compare the representations. PAA, APCA, PLA, PVQA, and SAX build their distance measures on the Euclidean distance [Keo97, YF00, KPMP01, LKLC03, MLW04]. The PAA distance measure is the distance between the mean values of all segments multiplied by the length of the segment:

$$d_{PAA}(\underline{\bar{y}}, \underline{\bar{y}}') = \sqrt{T/W}\sqrt{\sum_{w=1}^{W}(\bar{y}_w - \bar{y}'_w)^2} \tag{3.10}$$

The SAX distance measure is defined as the minimum distance between the symbols that represent a segment's mean value:

$$d_{SAX}(\underline{\hat{y}}, \underline{\hat{y}}') = \sqrt{T/W}\sqrt{\sum_{w=1}^{W} cell(\hat{y}_w, \hat{y}'_w)^2} \tag{3.11}$$

where

$$cell(a, a') = \begin{cases} 0 & |a - a'| \leq 1 \\ b_{max(a,a')} - b_{min(a,a')+1} & otherwise \end{cases} \tag{3.12}$$

For example, Figure 3.2 shows a time series $\underline{y}$ (blue line, $T = 4$) from [SK08]. Its PAA representation (red segments, $W = 4$) is $\underline{\bar{y}}^\intercal = (-0.70, -0.81, 0.08, 1.50)$. SAX visualizes the symbols with alphabetic characters ("a", "b", ...) in order to stress their discrete nature. Given an alphabet $A = 4$ and respective breakpoints at 0.00 and ± 0.67 (black horizontal lines and x-axis), its SAX representation is $\underline{\hat{y}}^\intercal = (a, a, c, d)$.

The figure shows a second time series $\underline{y}'$ (orange line) whose PAA and SAX representations are $\bar{\underline{y}}'^{\intercal} = (1.72, 0.34, 1.55, 0.49)$ (green segments) and $\hat{\underline{y}}'^{\intercal} = (d, c, d, c)$, respectively. The Euclidean distance between $\underline{y}$ and $\underline{y}'$ is approx. 6.71, the PAA distance between $\bar{\underline{y}}$ and $\bar{\underline{y}}'$ is approx. 6.44, and the SAX distance between $\hat{\underline{y}}$ and $\hat{\underline{y}}'$ is approx. 3.02.

These distance measures have the following properties.

- The distance calculation only requires W comparisons instead of T comparisons.
- PVQA and SAX use some storage to precalculate the distance. SAX stores the distance between all symbols as a lookup table of size A^2 so that d_{SAX} needs W lookups. Thus, it does not call the cell function (Equation 3.12) and it avoids the comparisons $min(a, a')$ and $max(a, a')$.
- The distance measures lower-bound the Euclidean distance, $d_*(y, y') \leq d_{ED}(y, y')$. Thus, the Euclidean distance is considered the baseline distance between two time series that is calculated approximately and efficiently by shape-based distance measures. For PVQA, the lower-bounding property has not been shown [MLW04]. By lower-bounding the Euclidean distance, shape-based distance measures fulfill an important requirement for time series matching.

Besides lock-step distance measures, techniques based on salient points apply *relative* distance measures [PWZP00, PF02] which relate the distance to the given values, and optimization techniques [MW01]. Further discretization techniques apply string matching as distance measure [AJB97, HY99]. Other techniques do not require a distance measure for their application [QWW98, BYS08, Fu11].

3.3 MODEL-BASED ENGINEERING

While raw-data-based and shape-based engineering focuses on the *effect* of a process expressed by a time series or its compressed shape, model-based engineering focuses on the *cause* of the process expressed by a generative *time series model*. This model consists of three components:

- The representation technique determines the class of the model which is assumed as cause of the time series.
- *Metaparameters* specify a model regarding important characteristics such as long-term or cyclically repeated characteristics.
- After identifying the representation technique and metaparameters, *model parameters* are estimated and form the *model-based* representation of a specific time series.

Model-based engineering follows a life cycle with five stages (Figure 3.3).

1. The *model identification* is the manual selection of a representation technique and of the manual or semi-automatic identification of metaparameters. Moreover, a distance measure is selected that expresses the distance of two representations based on model parameters.

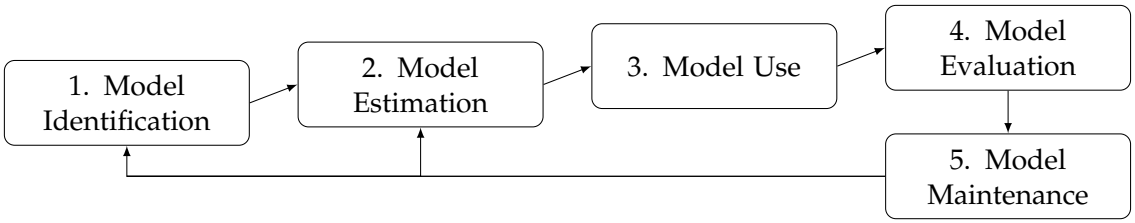


Figure 3.3: Life cycle of Time Series Model

2. The model parameters are *estimated* such that they represent a time series accurately.

3. Subsequently, the model parameters are *used* as model-based representation for a data-mining task.

4. *Model evaluation* provides diagnostic checks in order to re-evaluate the model.

5. *Model maintenance* uses the results from diagnostic checks to improve model identification and estimation.

Subsequently, we give an overview of model-based representation techniques from the literature along with distance measures and discuss their properties.

3.3.1 Representation

There are five classes of model-based representation techniques that we assess subsequently: *statistical models*, *ARIMA models*, *Markov models*, and *artificial neural networks*. Beyond, there are representation techniques such as Gaussian processes, time series bitmaps, or kernel models that have been applied rarely for more than one data-mining task which is why we do not include them in our assessment. More details on these techniques can be found in [MS82, Lia05, ASY15, BLB+17].

All these representation techniques are domain-independent since they focus on statistical causes in a time series. Domain-dependent models such as models for atmospherical phenomena [KKD91, BdMK02], are out of scope because they take physical information into account which is why they cannot be applied to others domains.

Statistical Model

A statistical model gathers the statistical assumptions concerning the generation of a time series. It treats values of a time series as realizations of a random variable Y and captures their distribution in a short representation. Formally, a statistical model is a pair $(\mathcal{S}, \mathcal{P})$ of the sample space $\mathcal{S}$ and a set of probability distributions $\mathcal{P}$. The sample space of a time series is the space of real numbers. The probability distributions are usually parametrized, i.e., $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ where θ is one set of model parameters from all possible sets of model parameters Θ. It is assumed that there is a true probability distribution that generates the time series values. The goal is to identify a set of probability distributions $\mathcal{P}$ and estimate model parameters θ such that P_θ approximates the true distribution [McC02].

For example, time series values are assumed to be normally distributed. The normal distribution $\mathcal{N}$ has two model parameters $\theta = (\mu, \sigma^2)$ where $\mu \in \mathbb{R}$ is the mean and $\sigma^2 \in \mathbb{R}_{>0}$ is the variance. Its probability density function is:

$$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(y-\mu)^2}{2\sigma^2}) \tag{3.13}$$

After the identification of normal distributions as underlying probability distributions $\mathcal{P}$, the model parameters have to be estimated to find $P_\theta \in \mathcal{P}$. This is carried out by a maximum-likelihood estimation. The likelihood function for the time series values y_1, ..., y_T is given by:

$$L(\mu, \sigma^2; y_1, ..., y_T) = \prod_{t=1}^{T} f_Y(y_t; \mu, \sigma^2) \tag{3.14}$$

$$= (2\pi\sigma^2)^{-T/2} \cdot exp(-\frac{1}{2\sigma^2} \sum_{t=1}^{T} (y_t - \mu)^2). \tag{3.15}$$

A logarithm is applied that leads to an easier calculation:

$$\log(L(\mu, \sigma^2; y_1, ..., y_T)) = -\frac{T}{2}\log(2\pi) - \frac{T}{2}\log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=1}^{T} (y_t - \mu)^2 \tag{3.16}$$

Finally, if the log-likelihood is maximized, it provides the estimations $\hat{\mu}$ and $\hat{\sigma}^2$ that approximate the true distribution:

$$\hat{\mu} = \frac{1}{T} \sum_{t=1}^{T} y_t \tag{3.17}$$

$$\hat{\sigma}^2 = \frac{1}{T} \sum_{t=1}^{T} (y_t - \hat{\mu})^2 = \left(\frac{1}{T} \sum_{t=1}^{T} y_t^2\right) - \left(\frac{1}{T} \sum_{t=1}^{T} y_t\right)^2 \tag{3.18}$$

Overall, the normal distribution represents a time series with two real values that can be calculated in one pass. When it comes to generating time series for meteorology, other probability distributions, such as Weibull or Reighley distributions are also assumed [KKSM91]. Moreover, Gaussian mixture models which are combinations of normal distributions have been applied to compute representations for audio signals [TW02].

ARIMA Model

The autoregressive integrated moving average (ARIMA) process provides a model for a variety of time series. It treats a time series as a realization from a stochastic process. As its name suggested, the model consists of an autoregressive (AR), a moving average (MA) part, and an integration (I) part that are derived from respective processes. Prerequisites of these processes are a white noise process and the linear process. First, we define these processes and second, we explain the parts of the ARIMA process.

White noise process An ARIMA process assumes that a time series is generated by a series of shocks which we call *error terms* a_t. These error terms are *independent* from each other, *randomly* drawn from a fixed distribution with mean 0 and variance σ_a^2, or more succintly $a_t \sim iid\, \mathcal{N}(0, \sigma_a^2)$ where iid is a shorthand for independent and identically distributed [SS11]. A series of these error terms $..., a_{t-1}, a_t$ is called *white noise process*. If the error terms are $a_t \sim iid\, \mathcal{N}(0, 1)$, then the series is called *normal* white noise process.

Linear process A process is called *stationary* if its probabilistic properties do not change over time and its values vary around a constant mean with a constant variance. A *linear process* assumes that a time series is generated by a weighted sum of error terms:

$$\tilde{y}_t = a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + ... \tag{3.19}$$

where ψ_1, ψ_2, ... are the weights and $\tilde{y}_t = y_t - \hat{\mu}$ is the time series corrected by its mean, if it is stationary. To be a valid stationary process, it is necessary for the weights to be *absolutely summable*, i.e., $\sum_{j=1}^{\infty} |\psi_j| < \infty$. Under suitable conditions, the linear process can also be regarded as weighted sum of past values of $\tilde{y}_t$ plus an added error term:

$$\tilde{y}_t = \sum_{j=0}^{\infty} \pi_j \tilde{y}_{t-j} + a_t \tag{3.20}$$

In this form, the process can be regarded as "regressed" on itself: y_t depends on former values plus noise.

Autoregressive process In practice, the representations of linear processes are not useful because they have an infinite amount of weights. The *autoregressive process* assumes a stationary linear process. However, the current value of an autoregressive process of order p, AR(p), is a finite, linear aggregate of previous p values:

$$\tilde{y}_t = \phi_1 \tilde{y}_{t-1} + \phi_2 \tilde{y}_{t-2} + ... + \phi_p \tilde{y}_{t-p} + a_t \tag{3.21}$$

where ϕ_1, ϕ_2, ..., ϕ_p are the weights, and a_t is an *error term* from a normal white noise process.

Using the backshift operator B which is defined as $By_t = y_{t-1}$ and the *operator* of AR(p) which is defined as:

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - ... \phi_p B^p \tag{3.22}$$

the autoregressive process may be written as:

$$\phi(B)\tilde{y}_t = a_t \tag{3.23}$$

Moving average process The *moving average process* is the second important stationary process. Instead on depending on former values of the process, a moving average process of the order q, MA(q), assumes a finite, linear aggregate q of error terms that occurred at time instance t and before:

$$\tilde{y}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q} \tag{3.24}$$

where θ_1, θ_2, ..., θ_q are weights. With the definition of an operator for MA(q):

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - ... \theta_q B^q \tag{3.25}$$

the moving average process is written as:

$$\tilde{y}_t = \theta(B) a_t \tag{3.26}$$

Mixed autoregressive-moving average process For some time series, it is necessary to combine an autoregressive and moving average process to provide an accurate model. The ARMA(p, q) process combines an AR(p) and a MA(q) process as follows:

$$\tilde{y}_t = \phi_1 \tilde{y}_{t-1} + ... + \phi_p \tilde{y}_{t-p} + a_t - \theta_1 \tilde{a}_{t-1} - ... - \phi_q \tilde{a}_{t-q} \tag{3.27}$$

or

$$\phi(B)\tilde{y}_t = \theta(B) a_t \tag{3.28}$$

ARIMA The autoregressive, moving average, and mixed autoregressive-moving average processes assume a stationary time series. However, some time series may exhibit a trend or a seasonal component which is why their values do not vary around a mean value. These time series can be transformed such that their differentiated representation is stationary. In case of a trend component, sometimes the dth difference of the time series is stationary. Thus it can be modeled with an ARMA process:

$$\phi(B)v_t = \theta(B)a_t \tag{3.29}$$

where

$$v_t = \nabla^d y_t = (1-B)^d y_t \tag{3.30}$$

The realizations y_t are a sum (or "integration") of v_t. Thus, an ARIMA(p, d, q) process is a sum of stationary ARMA(p, q) processes.

For a seasonal component, there also exists a differentiated representation taking into account the *season length* L:

$$\nabla_L y_t = (1-B^L)y_t = y_t - y_{t-L} \tag{3.31}$$

Here, the differentiation is carried out by subtracting the value of a season position from the value of the subsequent season position. For example, in a time series with monthly values, the value from Januar 2018 is subtracted from January 2019, February 2018 from February 2019 and so on. There are also higher order differentiations ∇_L^D which again differentiate these monthly differences. If an appropriate D has been identified, the time series does not exhibit seasonal behavior and it can be expressed as an ARIMA(P,D,Q) process:

$$\Phi(B^s)\nabla_L^D y_t = \Theta(B^s)\alpha_t \tag{3.32}$$

with P seasonal autoregressive weights Φ_1, ...Φ_P, Q seasonal moving average weights Θ_1, ..., Θ_Q, and error terms α_t. However, these error terms are correlated which is why a second non-seasonal ARIMA model (p, d, q) is introduced that corrects this:

$$\phi(B)\nabla^d \alpha_t = \theta(B)a_t \tag{3.33}$$

Finally, a seasonal ARIMA model is of order (p,d,q)x$(P,D,Q)_L$.

Based on the invertibility assumption [CP08], a seasonal ARIMA process can be represented by an *autoregressive expansion*, i.e., an AR(∞) process:

$$\pi(B)y_t = a_t \tag{3.34}$$

Thus, the weights π_1, π_2, ...may be used to represent a time series [Pic90]. It is proposed to truncate this sequence to at least $p+q+P+Q+1$ weights [CP08].

Although the representation size of ARIMA models is small, it is time-consuming to represent a time series with an appropriate ARIMA model. First, there is a large search space of metaparameters (p, q, d, P, Q, D) that have to be manually or semi-automatically identified. Second, estimating an ARIMA model usually involves optimization techniques that require many passes over a time series until the appropriate model parameters, i.e., weights π have been fitted.

Markov Model

Like an ARIMA model, a Markov model assumes that a time series is a realization of a stochastic process. However, while an ARIMA model assumes that a value y_t is a weighted combination of former values and error terms, a Markov model assumes the *Markov property*, i.e., a value y_t only depends on the predecessing value y_{t-1}.

The *Markov chain* is a Markov model that is applied in numerous works. It assumes that each individual time series value is discretized into a *state* $\hat{y} \in \{1, 2, ..., A\}$. A state is closely related to a symbol used in discretization (Section 3.2, page 26). However, a state uses an integer representation, i.e., $\{1, 2, ..., A\}$, while a symbol uses a character, i.e., $\{$"a", "b", ...$\}$. From one time instance to another, the state may change, i.e., there is a *state transition*. The Markov property suggests that the conditional probability of the current state only depends on the predecessing state:

$$p(Y_t = \hat{y}|(Y_1, ..., Y_{t-1})) = p(Y_t = \hat{y}|Y_{t-1}) \tag{3.35}$$

Thus, a Markov chain can be represented by the *transition probability matrix* P. It describes the conditional probability that a state j succeeds a state i: $p_{i,j} = p(Y_t = j|Y_{t-1} = i)$:

$$P = (p_{i,j}) = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,A} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,A} \\ \vdots & \vdots & \ddots & \vdots \\ p_{A,1} & p_{A,2} & \cdots & p_{A,A} \end{pmatrix} \tag{3.36}$$

An estimation of this matrix is calculated by counting the occurrences of state transitions in the time series. Let $n_i = |t : \hat{y}_t = i|$ be the absolute frequency, i.e., the number of occurrences, of state i in the time series and let $n_{i,j} = |t : 2 \leq t \wedge \hat{y}_{t-1} = i \wedge \hat{y}_t = j|$ be the absolute frequency of state transition i to j. Then $\hat{p}_{i,j} = n_{i,j}/n_i$ is the estimated conditional probability and $\hat{P} = (\hat{p}_{i,j})$ is the estimated transition probability matrix. Each row in the matrix represents a probability distribution since:

$$\sum_{j=1}^{A} \hat{p}_{i,j} = 1 \quad \text{for all } 1 \leq i \leq A \tag{3.37}$$

The representation is efficiently estimated in one pass. However, it consists of A^2 model parameters which is large.

Markov chains have been used for time series generation since the 1980s [JL86, KKSM91, SBW+05, PSAH15]. Few studies have been published on Markov chains for other data-mining tasks. One work presents their use for time series clustering [RSC02]. Further works focus on *hidden* Markov models (HMM) rather than Markov chains [Lia05, ASY15]. HMM are more complex and their representation is larger than the representation of Markov chains.

Artificial Neural Networks

Artificial neural networks (ANNs) are a family of models inspired by biological neural networks [GBC16]. They are universal approximators of functions with unknown type.

Artificial neurons are interconnected in order to transmit signals. The transmission activity depends on connection weights. A representation technique (perceptron, convolutional ANNs, recurrent ANNs [LBH15]) needs to be identified manually along with metaparameters, i.e., how many neurons are needed, how they are interconnected, and how they learn from each other. Model parameters are the weights that are estimated using given data.

Two data-mining tasks apply ANNs: time series generation and time series classification. While ANNs have been employed for time series generation, existing literature is scarce [APHRH13]. A recent review of the literature on time series classification found that ANNs were able to outperform raw-data-based techniques. However, they could not significantly outperform state-of-the-art shape-based techniques [IFW+19].

3.3.2 Distance

Statistical models express a time series as a probability density function. Applied to time series generation, they are used for evolving synthetic time series with same characteristics. Thus, two realizations of a statistical model, the given time series and the synthetic time series, are compared with each other [KKSM91]. Histograms plot their estimated probability distribution and show the frequency of realizations bucket-wise. However, literature on a numerical representation of this similarity is scarce.

ARIMA models are most often compared by the Euclidean distance of their truncated autoregressive expansion [Pic90, CP08, BJ14], some works focus on other distance measures such as a test-based similarity [Mah00] and a likelihood-based similarity [XY02].

Markov chains characterize a time series by a transition probability matrix. Each row in this matrix can be considered a probability distribution (Equation 3.37) and can be compared with the row from another transition probability matrix. The comparison of two estimated probability distributions is carried out by the Kullback-Leibler divergence:

$$d_{KL}(\hat{p}_i, \hat{p}'_i) = \sum_{j=1}^{A} \hat{p}_{i,j} \cdot \log \frac{\hat{p}_{i,j}}{\hat{p}'_{i,j}} \tag{3.38}$$

where $\hat{p}_i$ and $\hat{p}'_i$ are the i-th rows in two transition probability matrices $\hat{P}$ and $\hat{P}'$, respectively. To apply this distance to the full matrix, it is first symmetrized $d_{KL,s}(\hat{p}_i, \hat{p}'_i) = (d_{KL,s}(\hat{p}_i, \hat{p}'_i) + d_{KL,s}(\hat{p}'_i, \hat{p}_i))/2$ and subsequently, it is applied to all rows $d_{KL}(\hat{P}, \hat{P}') = \sum_i d_{KL,s}(\hat{p}_i, \hat{p}'_i)/A$ [RSC02].

Artificial neural networks use distance measures based on the Euclidean distance for comparing their outcome [APHRH13, MTV+17]. However, especially in time series classification, end-to-end architectures of ANNs are preferred which learn the data-mining task directly on a dataset without providing an intermediate representation [IFW+19].

Overall, ARIMA and ANNs require W comparison for the Euclidean distance calculation while Markov chains require $2 \cdot A^2$ for calculating the symmetric Kullback-Leibler divergence. None of these techniques require additional storage for the distance calculation and none of them focus on a lower-bounding property.

3.4 FEATURE-BASED ENGINEERING

Feature-based engineering is the second engineering class which compares time series based on their structure. Unlike model-based engineering it focuses on the *effect* of a time series process. A *feature* is a global time series property. It arises from the application of a method from the time-series analysis literature which characterises a time series as one real value [FLJ13, FJ14]. A *feature vector* gathers these characteristics and thus, represents a time series in a low-dimensional space. Subsequently, we present *feature-based* representations, related distance measures, and discuss their properties.

3.4.1 Representation

Time series features can be grouped into five categories, there are *distribution*, *correlation*, *component*, *stationarity*, and *frequency domain*. Subsequently, we give an overview of these categories along with example features. Beyond these five categories, there are further categories that describe time series [FLJ13]. There are several model-based representations among them whose parameters are regarded as features. Given that the focus is on an efficient and small representation, we do not include them in this work.

Distribution Features

Distribution features treat a time series as a set of values without focusing on the time domain. They provide summaries of these sets, such as moment features, location features as well as extreme and outlier value features [FLJ13]. Moment features, i.e., the *mean* and the *standard deviation* and the higher order moments *skewness* and *kurtosis* are applied in several works [TGDH93, NAM01, WSH06]. There are two versions of estimators that provide these features: the naive estimator and the unbiased estimator. Subsequently, we present the naive version since we work on long time series where this estimator is sufficiently close to the unbiased estimator.

The mean describes the center of the time series values, and is defined as follows:

$$m(\underline{y}) = \frac{1}{T} \sum_{t=1}^{T} y_t \tag{3.39}$$

The standard deviation is a measure of dispersion within the values (Equation 3.40). A low standard deviation means that the values are very narrow to the mean, whereas a high standard deviation represents a wide spread of the data. The variance is the squared standard deviation (Equation 3.41).

$$sd(\underline{y}) = \sqrt{\frac{1}{T} \cdot \sum_{t=1}^{T} (y_t - m(\underline{y}))^2} \tag{3.40}$$

$$var(\underline{y}) = \frac{1}{T} \cdot \sum_{t=1}^{T} (y_t - m(\underline{y}))^2 \tag{3.41}$$

$$= \frac{1}{T} \cdot \sum_{t=1}^{T} y_t^2 - \Big(\frac{1}{T} \cdot \sum_{t=1}^{T} y_t\Big)^2 \tag{3.42}$$

The skewness is the third standardized moment and represents the degree of asymmetry of the values around the mean. It is defined by:

$$skew(\underline{y}) = \frac{T^{-1}\sum_{t=1}^{T}(y_t - m(\underline{y}))^3}{(T^{-1}\sum_{t=1}^{T}(y_t - m(\underline{y}))^2)^{3/2}} \tag{3.43}$$

The skewness ranges around 0: $skew = 0$ means that the values are symmetric. If $skew < 0$, then the left tail of the distribution is longer, i.e., the values are skewed to the left. If $skew > 0$, then the right tail is longer, i.e., the values are skewed to the right.

The kurtosis is the fourth standardized moment. It represents the peakness or flatness relative to the probability density function of the normal distribution and is defined as:

$$kurt(\underline{y}) = \frac{T \cdot \sum_{t=1}^{T}(y_t - m(\underline{y}))^4}{(\sum_{t=1}^{T}(y_t - m(\underline{y}))^2)^2} \tag{3.44}$$

Kurtosis values range around 3: $kurt = 3$ means that the distribution of the values is as flat as the normal distribution, i.e., the tails of the distribution are as thin as the tails of the normal distribution. If $kurt < 3$, the distribution has a stronger peak and thinner tails. If $kurt > 3$, the distribution is more flat and has thicker tails.

While the mean and the variance/standard deviation (Equation 3.42) can be calculated in one pass, the skewness and kurtosis need two passes over the time series unless the mean is precalculated.

Correlation Features

The mutual dependence of a time series value to another is captured by correlation features [FLJ13]. The *autocorrelation* is frequently used as a feature [TGDH93, WSH06]. It expresses the linear dependence of a time series on itself, shifted by a sequence of lags. For example, a lag of 1 expresses the linear dependence between a time series with itself lagged by one time instance. It measures the linear predictability of a value y_{t+1} using only the value y_t and it is defined by:

$$acf_1(\underline{y}) = \frac{\sum_{t=1}^{T-1}(y_{t+1} - m(\underline{y}))(y_t - m(\underline{y}))}{\sum_{t=1}^{T}(y_t - m(\underline{y}))^2} \tag{3.45}$$

Autocorrelation values range between -1 and +1: $acf_1 = 0$ means that there is no linear dependence between a value y_t and its successor y_{t+1}. If $acf_1 > 0$, then there is a positive linear dependence between the values. The maximum correlation, $acf_1 = 1$ means that y_t perfectly predicts y_{t+1}. If $acf_1 < 0$, there is a negative linear dependence between successive values: if y_t increases, y_{t+1} decreases. This leads also to perfect predictability, if $acf_1 = -1$. Unless the mean is precalculated, this representation requires two passes over a time series.

Component Features

Several engineering techniques describe a time series as a combination of three components: a *trend*, a *season* and a *residual* component [SS11]. The trend represents the long-term change in the mean level of the series, whereas the season describes a cyclically

repeated behavior. Residuals represent unstructured information that is generally assumed to be random. The sum of these three components is called additive combination and represents time series from several domains:

$$\underline{y} = \underline{tr} + \underline{seas} + \underline{res} \tag{3.46}$$

where $\underline{tr}$, $\underline{seas}$, and $\underline{res}$ are the trend, season, and residual component, respectively. If a time series exhibits a multiplicative combination, it can be transformed into an additive combination by applying a logarithm. Components are extracted from a time series by *decomposition techniques* [HA13] and subsequently, they are reduced to features. Common features are the *trend* and *season strength* which express the influence of the respective component on the time series [WSH06, FLJ13]:

The trend strength represents the influence of the trend component on the time series (Equation 3.47). Its feature values range between 0 and 1: $R^2_{tr} = 0$ means that the series is determined by the residuals and the influence of the trend is negligible, whereas $R^2_{tr} = 1$ shows a high trend influence. These feature values also apply to the season strength R^2_{seas} which represents the influence of the seasonal component on the time series (Equation 3.48). Assuming the components have been extracted beforehand, the representation of the component strength needs one pass over the time series.

$$R^2_{tr}(\underline{y}) = 1 - \frac{var(\underline{res})}{var(\underline{res} + \underline{tr})} \tag{3.47}$$

$$R^2_{seas}(\underline{y}) = 1 - \frac{var(\underline{res})}{var(\underline{res} + \underline{seas})} \tag{3.48}$$

Stationarity Features

Stationarity is a notion of regularity which stems from model-based ARIMA representations (Subsection 3.3.1). It defines a time series as stationary if every set of time series values $\{y_{t_1}, y_{t_2}, ..., y_{t_K}\}$ has the same probabilistic properties as a set shifted by a lag h: $\{y_{t_1+h}, y_{t_2+h}, ..., y_{t_K+h}\}$. For example, when $K = 1$, y_{t+h} has the same probabilistic distribution as y_t which imposes that the mean of the time series is constant and does not depend on the time instance [SS11].

Feature-based representations do not assume a generative time series model and therefore, they express the stationarity of time series empirically. For example, the *StatAv* feature splits a time series into segments and calculates the mean in each segment which corresponds to a PAA representation. Then, it returns the standard deviation of the set of mean values. If the feature value is low it indicates that the time series is stationary, if it is high the time series is not stationary [PCH93]. One pass over a time series is required to calculate this feature. Further extensions are proposed that, for example, use sliding windows instead of fixed windows for calculating the mean values [FLJ13].

Frequency Domain Features

The aforementioned categories assume that features arise in the time domain. In some domains such as climate and physics, characteristics arise from periodic and sinusoidal variations. Therefore, time series are first transformed into the *frequency domain* and then,

these transforms are reduced to features. One example is the Fourier transform which transforms a time series into *Fourier coefficients*:

$$\hat{y}_f = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} y_t \exp(-\frac{2\pi i(f-1)(t-1)}{T}) \tag{3.49}$$

By selecting the most important coefficients, they may be used as features [AFS93] or they may be further filtered [FLJ13]. Besides the Fourier transform, also the wavelet transform is used for the representation. Both transformations are efficient as one Fourier coefficient requires one pass over the time series. However, if several coefficients are calculated, the Fourier transform requires several passes. A fast implementation requires $\mathcal{O}(T\ log\ T)$ operations [Mör03]. In contrast, a fast implementation of the wavelet transform requires only $\mathcal{O}(T)$ operations, which is more efficient [Mör03].

3.4.2 Distance

In most cases, the distance of feature representation is assessed by means of the Euclidean distance. This applies to time series matching and classification where Fourier coefficients represent time series [AFS93], and it applies to time series clustering where correlation features [GHLR01, WSH06] as well as distribution, and component features [WSH06] are selected. The frequency domain features provide the desirable property to lower-bound the Euclidean distance [AFS93, CF99]. Not all works need a distance measure for carrying out the data-mining task. Especially some classification techniques focus on a manual value comparison [PCH93], or they train an ANN [NAM01] or other classification algorithms [FJ14] directly on features.

3.5 SUMMARY

Table 3.1 summarizes the properties of all representation techniques and distance measures that have been presented. Recall that the variable T represents the time series length, W represents the amount of scalars in a representation, and A represents the alphabet size in case of a discretized representation. We make the following observations:

Representation Size Besides raw-data-based representations, the representation size of SAX and PVQA are the shortest compared to all other shape-based and model-based representations. Usually, W floating-point values need more storage than a discretized representation. For example, if SAX splits a time series into eight segments and discretizes them into an alphabet of size $A = 256$ each representation would occupy $64\ bit$. Another shape-based or model-based representation would occupy this storage with only two real values which is usually not an accurate representation. Feature-based representations are floating-point values, too. Even short feature vectors have a higher representation size than discretized shape-based representations.

Representation Time Most of the techniques need one or two passes to transform a time series into a representation. Many shape-based techniques require one pass over a time series. Sampling is the fastest representation technique, as it does not access every value. APCA adaptively determines the segment lengths and require many passes over a time series. Model-based techniques have different properties. While fitting a normal distribution is fast and requires only $64\ bit$, ARIMA models and ANNs need a time-consuming model identification and estimation. Markov chains represent a time series faster, but the representation size is quadratic in the alphabet size. Feature-based techniques only require one or two passes.

Distance Storage Raw-data-based, model-based, and feature-based techniques calculate the distance directly on the representations, as well as most of the shape-based techniques. In order to increase performance, PVQA and SAX precalculate distances once for the dataset and stores them in a lookup table. The additional storage is rather small; for a typical alphabet size of $A = 256$, the lookup table's size is approximatively $262\ kb$.

Distance Time While raw-data-based distance measures need T or T^2 comparisons, shape-based and model-based distance measures relying on the Euclidean distance measure only need W comparisons, which makes them much faster. As an exception, Markov chains need A^2 comparisons since they rely on a Kullback-Leibler divergence; whether this is faster or slower than the other approaches depends on the configuration of A and W. The distance calculation between two features is very fast. For short feature vectors, it can be even faster than the comparison of the other three engineering classes.

Lower-bounding Distance Some shape-based distances lower-bound the Euclidean distance (which also lower-bounds itself). The feature-based representation with Fourier coefficients also fulfills this requirement, other feature-based techniques do not.

We observe that feature-based engineering is a promising class of engineering techniques. It efficiently reduces time series to a set of real-valued features, which are efficiently compared, and it allows us to tackle the challenges of big time series datasets. More details will be given in the following chapter. Moreover, in Chapter 6, we will further investigate lower-bounding distance measures for our feature-based representation and on a discretized representation occupying even less space.

Table 3.1: Properties of Engineering Techniques

Class	Technique	Representation		Distance		
		Size (bit)	Time	Storage (32 bit)	Time	LB
Raw-data-based	ED	–	–	–	T	✓
	DTW	–	–	–	T^2	–
	cc	–	–	–	T	–
Shape-based	Sampling	$32 \cdot W$	$< \mathbf{1}$	–	W	–
	Extreme Points	$32 \cdot W$	1	–	W	–
	PAA	$32 \cdot W$	1	–	W	✓
	APCA	$64 \cdot W$	*	–	W	✓
	PLA	$64 \cdot W$	1	–	W	✓
	PVQA	$\mathbf{ld(A)} \cdot \mathbf{W}$	1	W	–	–
	SAX	$\mathbf{ld(A)} \cdot \mathbf{W}$	1	A^2	W	✓
Model-based	Normal Distr.	64	1	–	–	–
	ARIMA model	$32 \cdot W$	*	–	W	–
	Markov chain	$32 \cdot A^2$	1	–	A^2	–
	ANNs	$32 \cdot W$	*	–	W	–
Feature-based	Moments	32	$1-2$	0	1	–
	ACF1	32	$1-2$	0	1	–
	Component strength	32	2	0	1	–
	StatAv	32	1	0	1	–
	Fourier coefficient	32	1	0	1	✓

Representation time in passes over time series (asterisk "*" means many passes); Distance time in number of comparisons or lookups; LB means lower-bounding; A floating-point value occupies 32 bit

4

FEATURE-BASED ENGINEERING ACROSS DATA-MINING TASKS

FEATURE-BASED ENGINEERING is a promising technique as it tackles the challenges of big time series datasets by efficiently reducing time series to short vectors and comparing their similarity in structure. Many methods from the time-series analysis literature provide features, which leads to vast feature collections. A selection of important features is therefore required to keep the representation size short. Our goal is to select features that are applicable across data-mining tasks.

Many feature-based engineering techniques focus on specific data-mining task such as time series generation [KHSM17], time series matching [AFS93], time series classification [PCH93, NAM01, FJ14], and time series clustering [GHLR01, WSH06]. These techniques cannot be easily adopted by other data-mining tasks for two reasons: they do not have a specific property required by another data-mining task or, likewise, they rely on a specific property of the data-mining task they have been designed for. Time series matching, for example, requires engineering techniques with a lower-bounding distance measure. Techniques from other data-mining tasks do not have this property [PCH93, NAM01, WSH06, FJ14, KHSM17]. On the other hand, an engineering technique proposed by Fulcher et al. for classification contains thousands of features and thus, requires an automatical feature selection for training a classifier [FJ14]. Other data-mining tasks do not provide an automatical feature selection; thus, they cannot handle this representation efficiently.

Two feature-based engineering techniques from the frequency domain, the Fourier and wavelet transform, are applied to several data-mining tasks, including generation [CDB94], matching [AFS93, Mör03], classification, and clustering [Mör03]. However, they do not address the specific requirements of two data-mining tasks that we judge important: regarding generation, they do not simulate new stochastic characteristics, and regarding matching, they do not reduce the representation size to a discrete representation. Moreover, our time series also contain stochastic characteristics from the stochastic component, which blurs the representation in the frequency domain. Therefore, we do not select Fourier and wavelet transform as our engineering technique.

In this chapter, we propose a feature-based engineering technique based on features from the value and time domain, which is applicable to several data-mining tasks. We start by giving a design rationale explaining the idea behind our technique (Section 4.1). To support this design, we propose a suitable time series model (Section 4.2). The feature-based representation is retrieved by decomposing a time series into components and by

reducing the components to features. We select an existing decomposition technique and adapt it for our time series model (Section 4.3). Subsequently, we propose a representation based on distribution, correlation, and component features (Section 4.4) and introduce a suitable distance measure (Section 4.5). We summarize the advantages of our engineering technique over state-of-the-art techniques (Section 4.6).

4.1 DESIGN RATIONALE

It was decided that the best procedure for the design of an engineering technique is to study concrete big time series datasets from different domains. We choose three example datasets from the energy, meteorology, and economic domain that we detail subsequently.

Example 1 (Metering)**.** *The Irish Commission for Energy Regulation initiated the Smart Metering Project in order to assess the performance of smart meters in Ireland [The15]. The dataset contains the electricity consumption of approximately 6,000 households, small or medium businesses, as well as other entities between July 2009 and December 2010. The consumption has been measured in kilowatt-hour at a half-hour granularity. All recorded time series show seasonal components (daily, weekly, yearly) but lack a strong trend.*

Example 2 (Wind)**.** *The weather domain is represented by a dataset that contains time series of wind speed from 16 German airports between 2010 and 2016. The wind speed has been measured in meters per second at a half-hour granularity.*

Example 3 (Economy)**.** *The M3-Competition is the third of four M-Competitions [MH00]. Its goal is to compare forecast techniques on a defined dataset. The dataset contains about 3000 time series from different domains (finance, industry, demography, macro-economy, other). The values of each time series have a defined interval (year, quarter, month, other) and exhibit a trend and a seasonal component.*

Time series from the aforementioned domains can be long such as the Metering and Wind time series. Moreover, the Metering dataset contains approximately 6,000 time series which makes it large. We observe that the long-term behavior occurs in every value, and a cyclically repeated behavior reoccurs several times, which is why these time series can be reduced. We make the following observations that motivate our feature selection.

Observation 1: Features from the value and time domain The frequency domain captures characteristics of periodic and sinusoidal processes. Time series from the aforementioned domains exhibit such characteristics. However, they also contain stochastic characteristics that blur the representation in the frequency domain. Therefore, we focus on features from the value and time domain to characterize the time series.

Observation 2: Stationarity by decomposition Most time series from the aforementioned domains exhibit deterministic behavior. The wind speed is often stronger in winter than in other seasons which leads to yearly seasons. In long-term studies trends can also be observed. If human behavior comes into play, time series exhibit other season lengths. For example, weekly patterns can be observed in Metering due to a different behavior of consumers during weekdays and weekends. As a consequence, time series may have multiple seasons that should be taken into account. Economic time series exhibit long-term changes due to, e.g., an increase in sales of a product. As a consequence, we argue that extracting this behavior from time series is important for their characterization. The decomposition makes the residuals stationary which is why we do not need stationarity features for their characterization.

Observation 3: Deterministic components are highly predictable The trend and seasonal components provide a highly predictable behavior. Their characterization with distribution and correlation features would be meaningless because their value distribution is fixed and their autocorrelation is inherently high. Instead, we characterize them with component features.

Observation 4: Take stochastic patterns into account After extracting the deterministic components, the remaining residuals still have an important share in the overall signal. It is desirable to capture their characteristics, too. This finding is also confirmed in the literature. Theodosiou [The11] reports that autocorrelation remains in the residuals and should be taken into account. Modelers for time series generation report that generating wind speed with a Weibull distribution yields more realistic results than normal distribution because the given residuals are skewed and not symmetric [PSAH15]. The value distribution has been reduced to moment features by Nanopoulos in [NAM01]. The autocorrelation of lag 1 has been used as a feature for time series generation in [The11]. As a consequence, we argue that these distribution and correlation features should characterize residuals in our representation.

4.2 TIME SERIES MODEL

According to component features, a time series consists of a trend, a seasonal, and a residual component. Based on Observation 3, a time series can have several seasonal components with different season lengths that we take into account. We refer to the trend and seasonal components as *deterministic components*. Residuals are the *stochastic component* of a time series. Together, these components describe the *time series model* that we introduce subsequently. Formally, a time series is a combination of components:

$$\underline{y} = \underline{tr} + \sum_{s=1}^{S} \underline{seas}_s + \underline{res} \tag{4.1}$$

where $\underline{tr}$, $\underline{seas}_s$, and $\underline{res}$ are the trend, season, and residual component, respectively. The cyclically repeated characteristics of a season repeats T/L_s times where $L_s, 1 \leq s \leq S$ is the season length. The season length as well as the number of seasons S is fixed for every time series dataset. According to Equation 3.46 (page 38), we adopt an additive combination of components.

4.3 DECOMPOSITION

Knowing the components, we now look at decomposition techniques that can extract them. A decomposition does a non-unique split into the respective components, which can be further reduced to component features. We review existing techniques in Subsection 4.3.1 [KHL17b]. Then in Subsection 4.3.2, we propose our multiseasonal decomposition that we use for our time series model.

Table 4.1: Properties of Decomposition Techniques

	DEC	X-13	STL
Arbitrary season length	✓	-	✓
Decomposition of endpoints	-	✓	✓

4.3.1 Related Work

Regarding our design rationale, a decomposition technique should provide two important properties. As we observe a variety of season lengths (daily, weekly, yearly), it should handle season components of arbitrary season length. Moreover, it should decompose all time series values and not only a subset. There are several techniques based on moving-average filters or regression: *classical decomposition* and the more sophisticated techniques, *X-13* and *STL*. Subsequently, we review them regarding these two properties.

Classical decomposition dates back to the 1920s and is the basis for most subsequent decomposition techniques [KS83, HA13]. The key concept is to retrieve the trend by applying a moving-average filter on the given time series. Afterwards, the season is calculated by averaging the detrended values of associated time instances: in case of monthly values, all values of January are averaged, all values of February, and so on. The season may be of arbitrary season length. A major drawback is that this technique does not decompose the first and last values of the time series, called *endpoints*, due to the moving-average filter. Consequently, components cannot be completely retrieved.

In the 1970s, the X-11 technique from the U.S. Bureau of the Census was published and adopted by several statistical agencies around the world [Dag80]. This technique and its successors, X-12 and X-13, furthered the concept of classical decomposition with several moving-average filters [HA13, Bur17]. Most importantly, they use predictions from forecast models backward and forward in time such that the endpoints can be decomposed, too. However, the techniques are designed for decomposing only quarterly and monthly data. That is why this technique is not applicable to our general approach.

In the 1990s, Cleveland et al. [CCMT90] found that *Loess smoothing*, a locally-weighted regression technique, also leads to good results for detrending and deseasonalizing a time series. Their technique, STL, is considered a versatile and robust decomposition technique, handling every type of season length and decomposing endpoints [HA13]. It is widely and recently applied [The11].

Table 4.1 summarizes relevant properties of the presented techniques. STL fulfills these properties, which is why we adopt this technique for our approach.

4.3.2 Multi-seasonal Decomposition

STL handles arbitrary season lengths and decomposes the endpoints of a time series. But it only handles one seasonal component. To handle multiple seasonal components we adopt STL for our *multi-seasonal* decomposition technique, which splits a time series into components of our time series model (Equation 4.1). Multi-seasonal decomposition deseasonalizes a time series from the shortest up to the longest season. As input, a time series, a list of season lengths, and a list of *season granularities* are provided. A season granularity G_s determines the aggregation of time instances before the time series is deseasonalized. For instance, a yearly season is best extracted with monthly values, so the season length is 12, and the season granularity is "month".

Algorithm 4.1: Multi-seasonal Decomposition

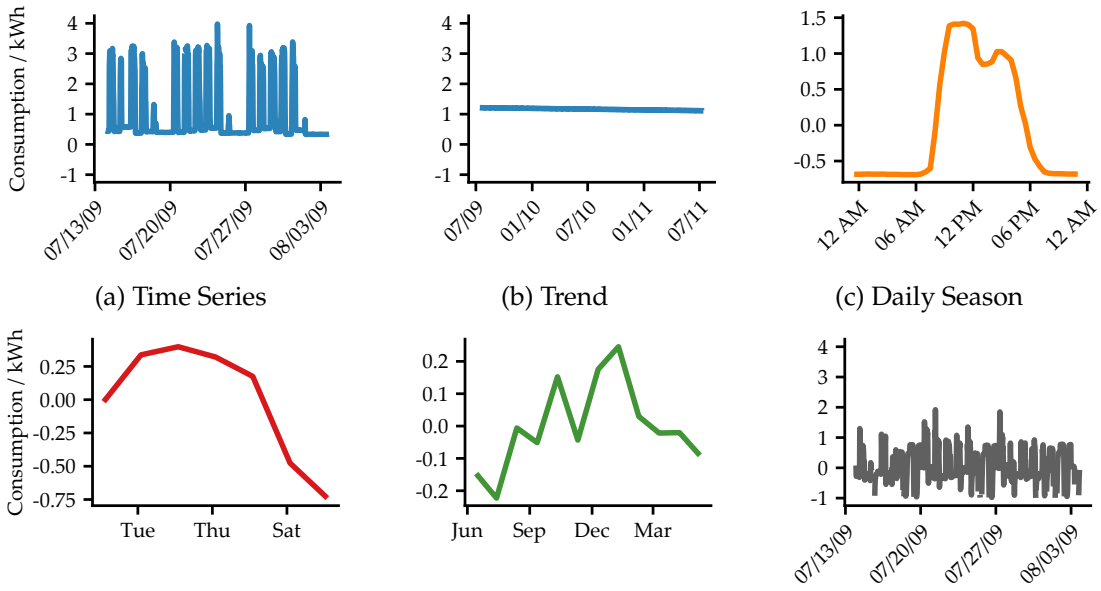
```
1  Sub ms_stl
2      remainder = y
3      res = y
4      For s = 1 To S
5          remainder = aggregate(remainder, G_s)
6          fit = stl(remainder, L_s)
7          remainder = remainder - fit.season
8
9          seas_s = disaggregate(fit.season)
10         res = res - seas_s
11     Next s
12
13     tr = disaggregate(fit.trend)
14
15     res = res - tr
16 End Sub
```

Algorithm 4.1 shows the pseudocode of multi-seasonal decomposition. The *remainder* (line 2) is the object that is successively decomposed. First, a loop (lines 4 – 11) extracts the seasons. The remainder is aggregated to the expected season granularity (line 5), STL decomposes the object into trend, season, and residuals (line 6). The season is extracted (line 7) and stored in the original granularity (line 9). Second, the trend component is the disaggregated trend fit (line 13). The residuals result as a subtraction of the extracted components and the given time series (lines 3, 10, and 15).

Figure 4.1 illustrates the multi-seasonal decomposition for a Metering time series. Figure 4.1a shows the first 20 days of the time series in a half-hourly granularity. Throughout the two years, we can observe only a slightly decreasing trend (Figure 4.1b). However, we can observe daily peaks in consumption, with less intensive consumption on the weekends. For the extraction of the daily season, we assume a half-hour season granularity (Figure 4.1c). A clear pattern with higher consumption during the day and a small peak at noon can be identified. We further aggregate the remainder to a daily granularity and extract the weekly season (Figure 4.1d). The pattern shows that consumption is higher during weekdays than the weekend. Finally, we aggregate the time series to a monthly granularity and extract the yearly season (Figure 4.1e). In order to extract the yearly season (which needs at least two cycles), it is assumed that the months January 2011 until June 2011 behave like January 2010 until June 2010. Due to the short time interval of two years, the monthly values are rather fluctuating. Nevertheless, they show an increased energy consumption during the winter months. An important share of the time series remains after extracting the aforementioned component. Figure 4.1f shows the first 20 days of these residuals. As expected, it contains less signal, and appears less systematical than the original time series.

4.4 FEATURE-BASED REPRESENTATION

Decomposition separates the deterministic and stochastic behavior of a time series and expresses it as components. However, components still have the same length as the original time series and they are not easier to handle. Therefore, they are reduced to features that we define as follows.



(a) Time Series (b) Trend (c) Daily Season

(d) Weekly Season (e) Yearly Season (f) Residuals

Figure 4.1: Multi-seasonal Decomposition

A feature is a mapping $f_k : \mathbb{R}^T \rightarrow \mathbb{R}$ that transforms a time series of length T into a scalar. A feature vector $\underline{f}^\intercal = (f_1, ..., f_k, ..., f_K)$ is a short representation with a *vector length* K, which captures the time series' important characteristics. Taking into account the aforementioned observations, we represent the deterministic components with component features (Subsection 4.4.1) and the stochastic component with distribution and correlation features (Subsection 4.4.2). Subsequently, we assess this feature-based representation regarding its size (Subsection 4.4.3) and time (Subsection 4.4.4).

4.4.1 Features for Deterministic Components

The trend component returned by STL contains the long term mean of the time series, *base value*, the long term change of this mean, *trend slope*, as well as local *trend changes*. In an attempt to identify trend features, we fit a linear regression model to the trend component:

$$\underline{tr} = \theta_1 + \theta_2 \cdot \underline{t} + \underline{\delta} \tag{4.2}$$

where θ_1 is the base value, θ_2 is the trend slope, and $\underline{\delta}$ is the vector of local trend changes. The vector $\underline{t}^\intercal = (1, ..., t, ..., T)$ is the vector of time instances of the time series. We use the base value and the trend slope as features and denote them $\theta_1(\underline{tr})$ and $\theta_2(\underline{tr})$, respectively. The trend changes are high-dimensional which is why we do not select them for our representation.

The season component is a cyclically repeated *season mask* that remains stable throughout the time series, which is configured by STL. The season mask is a set of L_s *seasonal features* $\sigma_l (1 \leq l \leq L_s)$ such that:

$$\sigma_l(\underline{seas}_s) = seas_{s,l} \tag{4.3}$$

for all $1 \leq l \leq L_s$. The season mask $\underline{\sigma}_s$ is in line with the season granularity G_s. Figures 4.1c - 4.1e show the season masks for the daily, weekly, and yearly season.

4.4.2 Features for Stochastic Component

Most of the meaningful information is extracted with the deterministic components. As observed in the design rationale, there is information remaining that cannot be described as a long-term change or a cyclically repeated behavior. These residuals are calculated by subtracting the deterministic components from the time series:

$$\underline{res} = \underline{y} - \underline{tr} - \sum_{s=1}^{S} \underline{seas}_s \tag{4.4}$$

We represent these residuals by distribution and correlation features: *standard deviation*, *skewness*, and *kurtosis* as well as the *autocorrelation of lag 1* (Equations 3.40, 3.43, 3.44 and 3.45, page 36f.). They are noted $sd(\underline{res})$, $skew(\underline{res})$, $kurt(\underline{res})$, and $acf_1(\underline{res})$, respectively. The mean of the residuals is assumed to be 0 since it is extracted with the trend component.

4.4.3 Representation Size

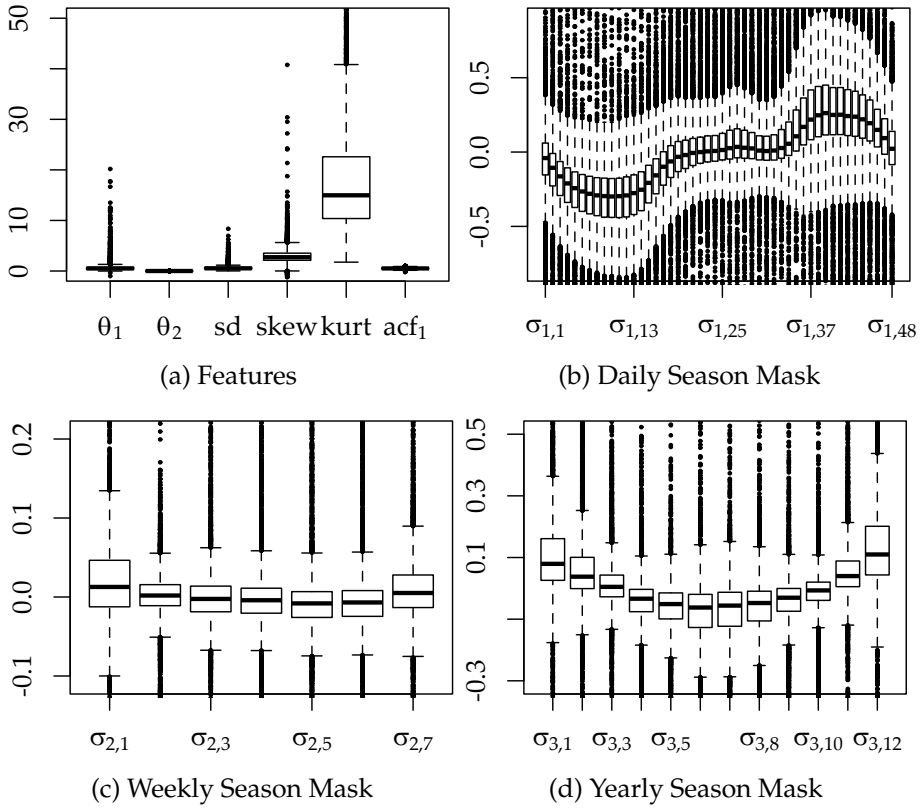
The features for the deterministic and stochastic components are combined in one feature vector:

$$\underline{f}^{\intercal} = (\theta_1(\underline{tr}), \theta_2(\underline{tr}), \sigma_1(\underline{seas}_1), \ldots, \sigma_{L_S}(\underline{seas}_S), sd(\underline{res}), skew(\underline{res}), kurt(\underline{res}), acf_1(\underline{res})) \tag{4.5}$$

Consequently, it reduces a time series to a representation of size $W = 6 + \sum_{s=1}^{S} L_S$. For example, a time series with a yearly season and a monthly aggregation level is reduced to $6 + 12 = 18$ features. The multi-seasonal time series from Figure 4.1 is represented with $6 + 48 + 7 + 12 = 73$ features. Still, this size is short compared to the raw-based representation (35,040 values).

The presented features are calculated for each time series of the Metering and the Wind dataset. Figure 4.2 presents the Metering features as boxplots. The features from the trend and stochastic component (Figure 4.2a) are heterogeneous, the share of extreme outliers (black circles) ranges between 1 and 8%. The season masks confirm this observation. The daily season of the Metering dataset is very fluctuating (Figure 4.2b) with a share of extreme outliers ranging between 4 and 11%. The season mask begins at midnight ($\sigma_{1,1}$) and finishes at 23:30 ($\sigma_{1,48}$). During the night ($\sigma_{1,1}$ until $\sigma_{1,12}$), the boxes indicate that less energy is consumed than during the day ($\sigma_{1,13}$ until $\sigma_{1,35}$). In the evening ($\sigma_{1,36}$ until $\sigma_{1,48}$), consumers tend to consume more energy. The weekly and yearly season masks (Figure 4.2c and Figure 4.2d) also indicate that the consumers are very heterogeneous as there are many outliers. A clear tendency of a weekly pattern, as for a single consumer (Figure 4.1), cannot be observed. However, a yearly seasonal behavior can be observed. The seasonal positions correspond to the months of the year, i.e., $\sigma_{3,1}$ is January, $\sigma_{3,2}$ February, and so on. In the summer term, there is clearly less energy consumed than in winter term. This fluctuation is mainly due to different consumption behavior resulting in differently shaped season masks.

Let us turn to the Wind dataset (Figure 4.3). The features of the Wind dataset have a small spread indicating a strong homogeneity (Figure 4.3a). The daily season of the Wind dataset has a clear shape (Figure 4.3c), as well as the yearly season (Figure 4.3b), indicating that there is more wind during the day than the night, and during winter than during summer. These observations are consistent with the meteorological literature [DBB+18].



(a) Features (b) Daily Season Mask

(c) Weekly Season Mask (d) Yearly Season Mask

Figure 4.2: Non-normalized Features (Metering)

Table 4.2: Mean Runtime per Time Series in Seconds (Metering)

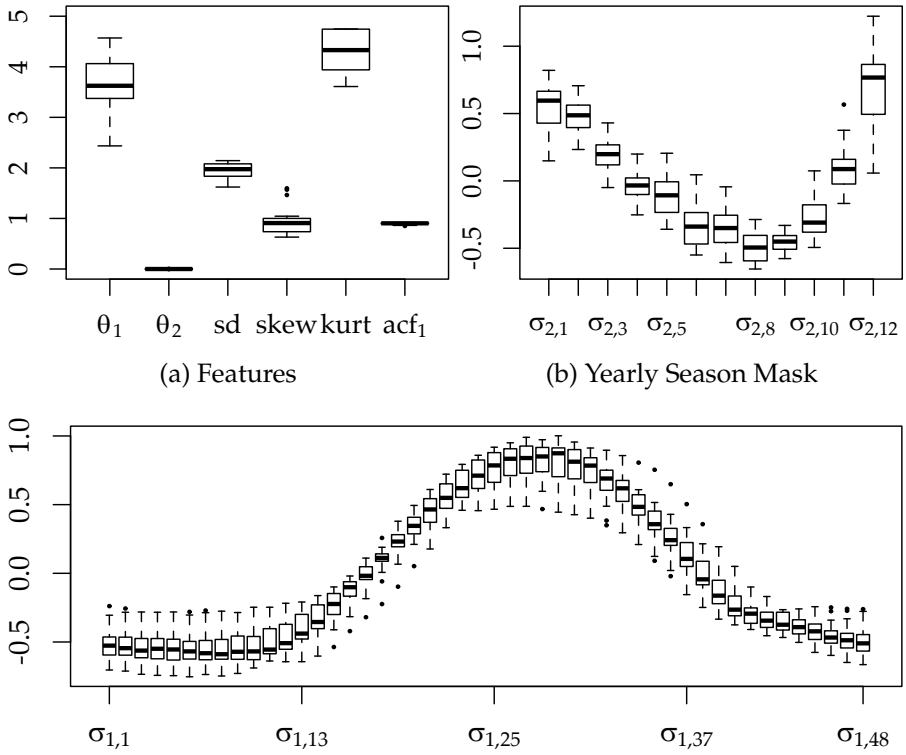
Season Length	T = 35,040		T = 350,400		T = 3,504,000	
	Dec.	Red.	Dec.	Red.	Dec.	Red.
48	0.049	0.013	0.497	0.132	6.170	1.460
480	0.039	0.013	0.396	0.151	4.534	1.576

Dec. means decomposition runtime; Red. means reduction runtime

4.4.4 Representation Time

We assess the representation time of our approach in order to assess its efficiency for big time series datasets. The representation time consists of the runtime for the multi-dimensional decomposition and the runtime for the feature reduction. We evaluate on the Metering dataset (2 years at half-hourly granularity) and measure the mean decomposition runtime and the mean reduction runtime per time series in seconds. Two dataset configurations are interesting for the approach because they influence the runtime of STL: the time series length T and the season length L_1. Other seasonal components (L_2 and L_3) are extracted on the aggregated time series, which is why they are much cheaper. We increase the time series length by a factor 10 and 100, leading to a length of 20 years ($T = 350,400$) and 200 years ($T = 3,504,000$). Moreover, we increase the daily season length L_1 by a factor 10, leading to time series of 73 days, 2 years, and 20 years at a 3-minute granularity, respectively.

Table 4.2 displays the mean runtime per time series by time series length and by season length. Although our representation technique requires several passes over a time series



(a) Features

(b) Yearly Season Mask

(c) Daily Season Mask

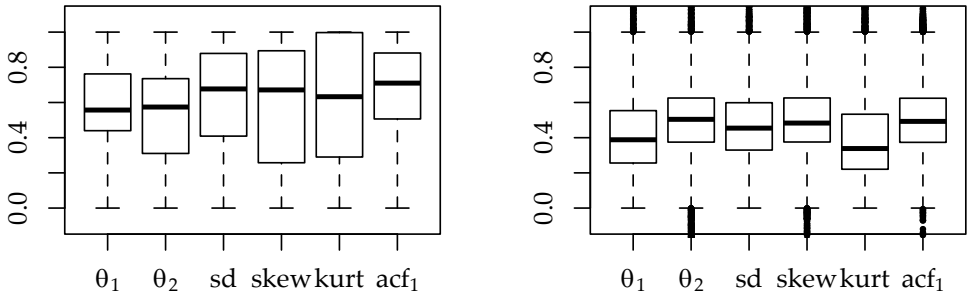
Figure 4.3: Non-normalized Features (Wind)

to decompose it and to reduce it into features, results suggest that our technique is very fast. Moreover, it scales linearly by the time series length, also representing very long time series with millions of values in an acceptable runtime. A longer daily season does not increase the decomposition runtime. Quite the contrary, it decreases it for two reasons: (1) at a higher granularity, the time series consists of less weeks or years, which need to be decomposed, and (2) the decomposition of the year is skipped if there are less than two years. The reduction runtime increases compared to a shorter season length since there are more seasonal features building up the feature vector. Overall, it takes approximately 6 minutes in total to represent the Metering dataset, which is acceptable for subsequent data-mining tasks.

4.5 FEATURE-BASED DISTANCE MEASURE

A feature-based distance measure compares two time series regarding their structural similarity. We define a distance measure that compares the selected features of two time series. It incorporates raw-data-based and feature-based distance measures to some extent: the deterministic features cover the shape of the time series while the stochastic features represent the value distribution and their autocorrelation.

To make features comparable, they have to be brought to a common range. Normalization has to (1) adjust feature values to a common range and (2) diminish the influence of outliers compared to the most frequent feature values. The boxplot provides an intuitive notion of common values (represented as box), near outliers (whiskers) and extreme outliers (black points), which we adopt for our *0-1-normalization with outliers* as follows.



(a) Wind (b) Metering

Figure 4.4: Normalized Features

Let $F = \{f(\underline{y}_i), 1 \leq i \leq I\}$ be the values that a single feature exhibits in a time series dataset. Let $Q_1(F)$ be the value of the lower quartile and $Q_3(F)$ the value of the upper quartile, respectively. Let $IQR(F) = Q_3(F) - Q_1(F)$ be the interquartile range. We define a *normalized* feature $f^{01out}(\underline{y})$ based on:

$$\begin{aligned} lower(F) &= \arg\min_{f(\underline{y}) \in F} \{f(\underline{y}) \geq median(F) - 1.5 \cdot IQR(F)\} \\ upper(F) &= \arg\max_{f(\underline{y}) \in F} \{f(\underline{y}) \leq median(F) + 1.5 \cdot IQR(F)\} \end{aligned} \tag{4.6}$$

as follows:

$$f^{01out}(\underline{y}) = (f(\underline{y}) - lower(F))/(upper(F) - lower(F)) \tag{4.7}$$

where f^{01out} is the feature that ranges between 0 and 1 except for extreme outliers. Figure 4.4 illustrates how features are shifted to the same range. The values we are focusing on, are between 0 and 1, the extreme outliers are below or above this range. Their share has not changed compared to the non-normalized representation. Based on the normalized features, we now define the *feature-based distance measure* that assesses the similarity of two time series. Let $\underline{y}$ and $\underline{y}'$ be two time series. The feature-based distance d_k of the feature f_k is given by Equation 4.8. Moreover, the *overall feature-based distance* is the RMSE of all feature-based distances (Equation 4.9). This distance is either close to 0, i.e., time series are highly similar, between 0 and 1, i.e., the features of time series are in the common range or close to it, or greater than 1, i.e., some features are not in the common range.

$$d_k(\underline{y}, \underline{y}') = \mid f_k^{01out}(\underline{y}) - f_k^{01out}(\underline{y}') \mid \tag{4.8}$$

$$d_f(\underline{y}, \underline{y}') = \sqrt{\frac{1}{K} \sum_{k=1}^{K} d_k(\underline{y}, \underline{y}')^2} \tag{4.9}$$

4.6 SUMMARY

Our feature-based engineering technique provides promising properties compared to other engineering techniques. The feature vector size is $W = 6 + \sum_{s=1}^{S} L_S$ and thus, the representation of one time series occupies $32 \cdot W$ *bit*. This is comparable to the size of some shape- and model-based representations. The technique provides a representation in a reasonable time, as our evaluation suggests. Big time series datasets can be efficiently represented in a few minutes, although STL needs several passes. The distance measure calculates the distance directly on the features. It requires no additional storage and it calculates the result with W comparisons which, again, is comparable to state of the art. However, this distance measure is not lower-bounding because it does not provide an estimation of the Euclidean distance. We defer this discussion to Chapter 6 where we present a discretized feature-based representation calculated in up to two passes.

5

TIME SERIES GENERATION

FEATURE-BASED ENGINEERING has several advantages over other engineering classes, as discussed in Chapter 3. It efficiently calculates a short time series representation that can be compared efficiently regarding the similarity in structure. Our feature-based engineering technique proposed in Chapter 4 provides these properties for several domains. Let us now turn to the second part of this work: we apply this technique to data-mining tasks and begin with time series generation. Time series generation evolves a generated time series dataset from a given dataset. It is a crucial data-mining task for two major reasons: system evaluation and data availability.

System evaluation is a crucial phase in many industrial and organizational processes. It is done to test and verify component performance, to assess system robustness, and to estimate the correct sizing of necessary resources. Generated datasets are vital to this phase by providing comparability and a variety of possible inputs for the systematic and thorough assessment of a system [JL86, CDB94, SJ13]. Also, generated datasets can include user-given hypotheses. Thus, they allow users to study system behavior and assess risks in possible future scenarios [vPL02, KHL17b].

Data availability covers the many side conditions that accompany data gathering in general. Often, data might not be available in the required amount or quality, due to very complex and expensive measurement procedures or because of sensitive and error-prone sensors. In addition to these technical issues, legal regulations often restrict data availability by concerning privacy and intellectual property. With generated datasets, the impact of these issues can be lessened, e.g., by compensating missing or erroneous data and by anonymizing confidential data [KKD91, BdMK02, MH15, ILD+17].

Generation techniques are applied in a multitude of domains. As reported in Chapter 2, they are used in the meteorological domain, where a statistical model was first mentioned in 1982 for the generation of wind speed time series [MS82]. Since then, researchers have been developing further generation techniques for simulating weather parameters such as wind speed and direction, solar radiation, humidity, and precipitation [JL86, KKD91, BdMK02, MH15]. In the energy domain, generated data assess renewable energy power plants [JL86, ILD+17], while in industry and computing, they evaluate various purposes [CDB94, SJ13].

While these techniques are valuable, they represent isolated solutions tailored to their specific application; thus, they only cover specific characteristics. Concerning the importance of time series generation, this poses a challenge as it makes the topic difficult

to access for new users that want to utilize it in their domain. In the worst case, an interested user would have to survey existing work before either adopting a technique to his/her respective domain or starting from scratch.

Our first objective is a comparative and domain-independent assessment of generation techniques. To compare what generation techniques can express, we apply a feature-based distance since it captures well the important characteristics of given datasets. Our second objective is a generation technique that is applicable across domains. We use the feature-based representation to capture the important characteristics of a given dataset and to evolve a new dataset that is highly similar to the given dataset.

This chapter is structured as follows. We review existing generation techniques in Section 5.1 together with distance measures which commonly assess generation techniques. In Section 5.2, we introduce and discuss two feature-based generation techniques. They not only allow the controlled generation of time series that abide by the specific characteristics of their originating domain but also provide a way to evolve new characteristics for hypothetical "what-if" scenarios. We compare the performance with existing generation techniques in Section 5.3 before we summarize our results in Section 5.4.

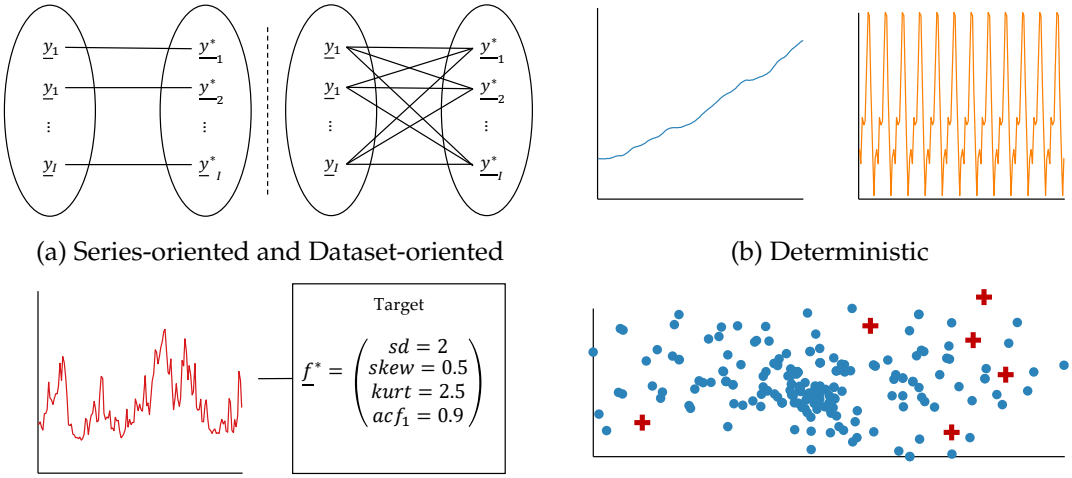
5.1 STATE OF THE ART

We review generation techniques for time series from the literature and compare them regarding their class, their properties, and their expressiveness. For this review, we only select domain-independent techniques. Domain-dependent techniques are out of scope because they take physical information of a given time series into account, and thus, they cannot be applied to other domains. For example, there are techniques for wind power generation, taking into account the specific power curve of a wind power plant, including cut-in and cut-off speed. Clearly, such techniques cannot be adopted by other domains. Finally, we summarize distance measures that commonly assess the accuracy of the generated time series.

In the literature, there are *raw-data-based* generation techniques and *model-based* generation techniques. The former take a given time series dataset as input while the latter take a model-based representation of a dataset as input. We review them in Subsections 5.1.2 and 5.1.3, respectively. Beyond this classification, we characterize generation techniques regarding four principal properties. A technique should fulfill all these properties to be comprehensive and widely applicable. We present these properties in Subsection 5.1.1 before reviewing the generation techniques. Finally, we review generation techniques regarding their expressiveness, i.e., the time series characteristics they can capture and reproduce. Subsection 5.1.4 gives an overview of characteristics that are important for time series generation across domains. In Subsection 5.1.5, we compare the generation techniques regarding their properties and their expressiveness.

5.1.1 Properties of Generation Techniques

As illustrated in Figure 5.1, we characterize generation techniques with four properties.



(a) Series-oriented and Dataset-oriented

(b) Deterministic

(c) Stochastic

(d) Innovative

Figure 5.1: Properties of Generation Techniques

Dataset-oriented There are two different input scopes for a generation technique: either it focuses on one time series at a time or on the whole dataset at a time (Figure 5.1a). In the former case, the technique reproduces the characteristics of only one time series while ignoring the characteristics from other time series of the dataset. Dataset-oriented techniques take advantage of all characteristics that arise in a dataset, and they can take relationships of time series into account [ILD+17, KHSM17].

Deterministic A deterministic generation technique reproduces deterministic characteristics of a time series which arise from a trend or seasons (Figure 5.1b). These characteristics are shuffled or recombined [SJ13, ILD+17].

Stochastic A stochastic generation technique captures the stochastic characteristics of a time series, i.e., distribution and correlation characteristics (Figure 5.1c). Moreover, these characteristics are used to simulate new random values instead of taking the randomness as is [KKSM91, PSAH15].

Innovative There are techniques that not only reflect given characteristics but also incorporate new characteristics in a generated dataset. A *feature space* (Figure 5.1d) visualizes the characteristics or features of a dataset in a scatter plot [KHSM17]. Blue dots represent given time series, while red crosses represent time series evolved by a generation technique. This generation technique is innovative because evolved time series have new characteristics not covered by the given dataset. Such innovative techniques are important when it comes to inflate a dataset or to provide "what-if" scenarios [KHL17a, KHL17b, FPD+17].

Generation techniques that fulfill all these properties are considered comprehensive since they can evolve time series for a variety of domains. Subsequently, we present generation techniques regarding these properties and summarize them in Table 5.1.

5.1.2 Raw-data-based Generation Techniques

Raw-data-based generation techniques do not need an intermediate representation and generate a dataset directly on the given dataset. They are based on *bootstrapping*, *averaging*, *recombination*, or on a *genetic algorithm*.

Bootstrapping

Bootstrapping splits a time series into intervals of the same length and permutes the values within these intervals [SJ13]. The technique can generate several new time series from one given time series. Obtained results stay similar to the original data as long as the chosen interval length is reasonably small. We conclude that bootstrapping is a deterministic but not stochastic technique. It only focuses on one time series at a time, and it is not innovative.

Averaging

Averaging evolves new time series by averaging given time series from a dataset. It can be combined with varying weights to increase variety and in order to follow the characteristics of the given data [FPD+17]. This technique is dataset-oriented, deterministic, and innovative. However, since distribution and correlation characteristics are taken as is, we consider this technique non-stochastic.

Recombination

The recombination technique first decomposes the values of time series into a trend, seasons, and residuals. Then it shuffles these components and finally recombines them in a new order. Time series generated in that way keep the characteristics of the original dataset but in new combinations. Iftikhar et al. [ILD+17] brings time series recombination together with ARIMA models. The authors cluster components and shuffle them within their clusters. New residuals are simulated using an autoregressive model.

Overall, recombination is a dataset-oriented and deterministic generation technique. It is innovative in that it creates new combinations of characteristics. However, it is only partially considered stochastic as the randomness stems from an ARIMA model [ILD+17] and does not take distribution characteristics into account.

Genetic Algorithm

The genetic algorithm generates time series by combining randomly selected given time series and ensures that the result is as close as possible to a given set of characteristics. A combination is carried out by selection, crossover, and mutation processes that occur with a specified probability. This technique, which was presented by Kang et al. [KHSM17], is considered a dataset-oriented and deterministic generation technique. It is innovative in that it can evolve new characteristics. Mutation evolves new random values even though they are not modeled.

5.1.3 Model-based Generation Techniques

Model-based generation techniques use a model-based representation as an intermediate. We could identify four representations in the literature: *statistical models*, *ARIMA models*, *Markov chains*, and *artificial neural networks*, which we introduced in Section 3.3 (page 29). Subsequently, we explain how they are used for generating new time series.

Statistical Model

As mentioned earlier, statistical models gather statistical assumptions of a time series in a model that needs to be identified and whose parameters need to be estimated. These models can be used for evolving new time series. One example is a random variable that generates a sample of independent and identically distributed values with a specific value distribution such as a normal distribution. While there is a plethora of value distributions, literature regarding generation is sparse. Besides the normal distribution, authors used the Weibull, and Reighley distribution [KKSM91].

Usually, these samples are not immediately obtained because there is no direct sample generator for each value distribution. However, a sample of each (continuous) random variable Z can be generated from a uniformly distributed sample using *inversion sampling* [Kol08]. Let $F_Z : \mathbb{R} \to [0, 1]$ be the cumulative distribution function of Z, and U a random variable that is uniformly distributed between 0 and 1. Inversion sampling states that the random variable Z can be obtained from the inverse of its cumulative distribution function $Z = F_Z^{-1}(U)$. This means, a sample u from U can be transformed into a sample z from an arbitrary (continuous) random variable Z. These samples are the time series values with the expected value distribution.

In summary, statistical models are stochastic but they are not deterministic, dataset-oriented, or innovative.

ARIMA Model

Besides the specific value distribution, the correlation between the values of a time series is also important. Inversion sampling cannot achieve this. Instead, ARIMA models are employed for this characteristic. Recall that an $AR(p)$ model is of the form $\phi(B)\tilde{y}_t = a_t$ where $\phi(B)$ are the weights using the backshift operator, and a_t is an error term from a normal white noise process. The correlation can be calculated from the weights such that a simulation of an autoregressive model exhibits an expected autocorrelation in the generated time series. Specifically for an $AR(1)$ model, the autocorrelation of lag 1 equals ϕ_1 [SS11].

After model identification and estimation, the ARIMA model is used for time series generation. First, a normal white noise process is simulated. Second, these simulations are fed into the model leading to a generated time series incorporating the target correlation characteristics. The generated values are normally distributed, like the white noise process. If the given time series exhibits a value distribution other than a normal distribution, it may be poorly expressed in a generated time series.

ARIMA models can also include deterministic characteristics from a given time series [HK08]. In summary, they are deterministic and stochastic but not dataset-oriented or innovative.

Markov Chain

A Markov chain is a technique reproducing both correlation and distribution characteristics. It represents a time series as a transition probability matrix $\hat{P} = (\hat{p}_{i,j})$ where $\hat{p}_{i,j}$ is the estimated conditional probability that a state j succeeds a state i. The generation is carried out by simulating the transitions with a sample of uniformly distributed values

between 0 and 1. For example, let us assume that a Markov chain consists of two states ($A = 2$) and the estimated probabilities of state 1 are $\hat{p}_{1,1} = 0.5$ and $\hat{p}_{1,2} = 0.5$. If the sample value is smaller or equal 0.5, then a state transition from 1 to 1 is simulated, if the sample value is greater than 0.5, then a state transition from 1 to 2 is simulated. Before the simulation of state transitions, the first state of a time series has to be determined. Either the first state of the given time series is used as the first value of the generated time series [PSAH15], or the first state is selected randomly [SBW+05]. Other works do not discuss how they determine the first state [JL86, KKSM91]. Finally, the resulting vector of states is transformed back to real values. Either one representative value from the state interval is selected [JL86] or a uniform distribution generates a real value from the state interval [SBW+05, PSAH15].

This generation technique can reproduce the distribution and correlation characteristics of a time series pretty well [BK09]. It has been used since the 1980s [JL86]. Since then, it was extended to better capture the autocorrelation of longer lags which is why Markov chains of second order were applied [KKSM91]. More recently, Pesch et al. focus on finding the best distance for the second lag in order to optimally fit a Markov chain to a given scenario [PSAH15]. The problem of Markov chains of orders higher than 2 is their increased representation size, therefore existing research is generally limited to this order. In summary, this generation technique is not dataset-oriented because it trains one transition probability matrix per time series. It is stochastic but not deterministic or innovative.

Artificial Neural Networks

Artificial neural networks (ANNs) are universal approximators of functions with unknown type. They have been applied for time series generation, however existing literature is scarce. Almonacid et al. use a multilayer perceptron [APHRH13]. They feed in characteristics from a time series as input and expect that the network returns the time series as output. As such, they train the network on a given dataset. Subsequently, they evolve new time series by feeding in new combinations of characteristics.

Regarding this reference, an ANN is dataset-oriented, deterministic, and innovative. While ANNs in general are capable of evolving stochastic characteristics, the present technique is not. Therefore, we classify it as non-stochastic.

5.1.4 Assessing Expressiveness

To assess the expressiveness of a generation technique, a generated dataset is compared to a given dataset using a distance measure. There are *numerical* distance measures from different engineering techniques (Chapter 3) as well as *visual* distance measures, which we summarize subsequently.

Most often, a raw-data-based distance measure is assessed, either visually by providing line plots or scatter plots [KKD91, BdMK02, APHRH13, ILD+17, KHSM17, CDB94, BK09, PSAH15], or numerically [BdMK02, APHRH13, BK09, SJ13]. The root mean squared error measure (RMSE) is the most frequent numerical distance measure. It is a lock-step distance measure based on the Euclidean distance (Equation 3.3, page 25). It represents the mean distance between the values of the given and the generated time series, whereas higher distances are more influential due to squaring.

Table 5.1: Properties of Generation Techniques

	BT	AV	RE	GA	SM	AM	MC	ANN
Dataset-oriented	–	✓	✓	✓	–	–	–	✓
Deterministic	✓	✓	✓	✓	–	✓	–	✓
Stochastic	–	–	(✓)	(✓)	✓	✓	✓	–
Innovative	–	✓	✓	✓	–	–	–	✓

Statistical Model (SM); ARIMA Model (AM); Markov Chain (MC); Artificial Neural Network (ANN); Bootstrapping (BT); Averaging (AV); Recombination (RE); Genetic Algorithm (GA)

Raw-data-based distance measures focus on the time domain, by comparing values at each time instance. In contrast, *distribution distance measures* focus on the value domain rather than the time domain. They assess the distribution of time series values independent from their time instance. Histograms are an appropriate visual distance measure for this class: they plot the estimated probability distribution of given and generated time series values and show the frequency of occurrences bucket-wise [KKSM91, JL86, BdMK02, APHRH13, PSAH15]. Thus, two time series are compared using the frequency of their values. However, the literature on generation techniques that numerically assess this distance is scarce. For Markov chains, it can be shown that the generated time series nearly follows the probability distribution of the given time series [BK09]. However, the comparison of the distribution with a *histogram distance measure* or the comparison of distribution features (Section 3.4, page 36) is not carried out. While distribution features would compare the moments of two distributions, this histogram distance measure would compare two distributions bucket-wise.

Finally, the *autocorrelation distance measure* often assesses the correlation of a time series [JL86, KKD91, KKSM91, BdMK02, BK09, PSAH15]. It is visualized by a line plot whose x-axis is the sequence of lags and whose y-axis is the corresponding autocorrelation. Thus, the autocorrelation of a generated time series is similar to a given time series if their lines are narrow to each other. Numerically, these lines are assessed by the RMSE [BK09].

Several other characteristics may be assessed with a distance measure. However, we focus on the presented ones since they are most general, and they are frequently employed to assess generation techniques from different domains.

5.1.5 Comparison

We compare the reviewed generation techniques regarding the four properties: dataset-oriented, deterministic, stochastic, and innovative. If a generation technique fulfills all these properties, it is considered comprehensive, as it can evolve time series for a variety of domains. However, as Table 5.1 shows, none of generation techniques fulfill all the properties to a full extent. Subsequently, we establish generation techniques addressing this issue.

Most often, raw-data-based, distribution, and autocorrelation distance measures assess the expressiveness of generation techniques. Throughout this chapter, we use the term *standard distance measure* to refer to them, and we assess them visually and numerically. We hypothesize that the feature-based distance measure describes them well: by comparing the deterministic features, it assesses the overall raw-data-based distance measure and by comparing the distribution and autocorrelation features, it assesses the histogram and autocorrelation distance measure. Moreover, it provides insights which features are well reproduced and which features are not.

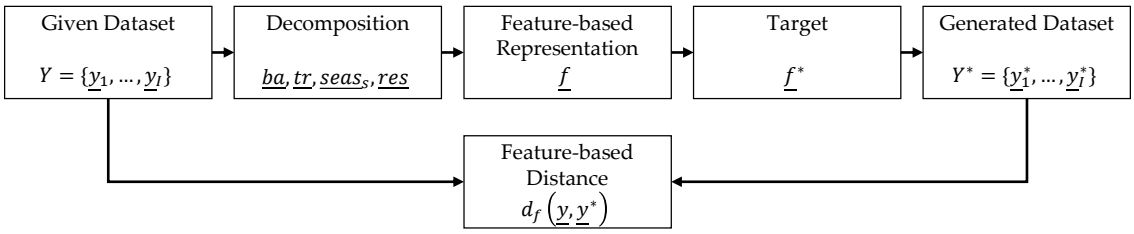


Figure 5.2: Overview of Feature-based Generation

5.2 FEATURE-BASED GENERATION

In this section, we present two feature-based generation techniques, a technique based on *modification* and a technique based on *recombination*. The common steps of these techniques are highlighted in Figure 5.2. After retrieving a *given dataset* each time series is *decomposed* and reduced to a *feature-based representation*. A *target* feature vector $\underline{f}^*$ indicates for which features a time series shall be generated. Generation techniques provide a generated time series $\underline{y}^*$ for this target, which results in a *generated dataset* Y^*. This dataset can be compared with the given dataset by the *feature-based distance* for assessing their similarity. Thus, features fulfill two tasks: (1) they provide a characterization of a given dataset that can be employed for generation and (2) they assess the expressiveness of generation techniques. Subsequently, we present our two feature-based generation techniques based on modification (Subsection 5.2.1) and recombination (Subsection 5.2.2) [KHL17a, KHL17b, KHL18].

5.2.1 Feature-based Modification

Our first generation technique relies on the observation that time series components can be modified for evolving new time series. We propose to modify components by introducing *factors* and to calculate these factors such that the new time series agrees with a target feature vector.

Modification with Factors

We describe three factors β, γ, and κ that express the modification of a time series component and that affect the three features trend strength (Equation 5.1), trend slope (Equation 5.2), as well as season strength (Equation 5.3). Recall that these features are calculated as follows:

$$R^2_{tr}(\underline{y}) = 1 - \frac{var(\underline{res})}{var(\underline{res} + \underline{tr})} \tag{5.1}$$

$$\theta_2(\underline{tr}) = \theta_2 \text{ where } \underline{tr} = \theta_1 + \theta_2 \cdot \underline{t} + \underline{\delta} \tag{5.2}$$

$$R^2_{seas}(\underline{y}) = 1 - \frac{var(\underline{res})}{var(\underline{res} + \underline{seas})} \tag{5.3}$$

Subsequently, we introduce the three factors that express how a time series component is modified. Figure 5.3 visualizes modifications on the Economy and Metering dataset.

Trend Strength Factor Let β be a factor that varies the trend strength:

$$\underline{tr}^* = \theta_1 + \beta \cdot (\theta_2 \cdot (\underline{t} - 1) + \underline{\delta}) \tag{5.4}$$

This equation represents the linear regression model that is fitted to the STL trend. The factor β is applied to the trend slope θ_2 and the trend change $\underline{\delta}$. Depending on β, the trend strength increases ($\beta > 1$), decreases ($0 \leq \beta < 1$), or is left unchanged ($\beta = 1$). A factor $\beta < 0$ is not admissible.

The effect of this factor is represented by Figure 5.3a on a trend component from the Economy dataset. The plot shows the original trend (blue with triangles) and three modified trends. The latter ones are modified by a trend determination factor $\beta = 1.25$, $\beta = 0.75$, and $\beta = 0.50$, respectively. Overall, the important characteristics of the trend are kept but they are a multiple of the former value. The influence on the trend strength R^2_{tr} is given in the figure's legend.

Trend Slope Factor Let γ be a factor that varies the trend slope:

$$\underline{tr}^* = \theta_1 + \gamma \cdot \theta_2 \cdot (\underline{t} - 1) + \underline{\delta} \tag{5.5}$$

Again, we apply the factor to the linear regression model. But in this case, only the trend slope is modified and not the trend change. Depending on γ, θ_2 increases ($\gamma > 1$), decreases ($0 \leq \gamma < 1$), or is left unchanged ($\gamma = 0$). $\gamma < 0$ is not admissible.

This effect is represented in Figure 5.3b on the aforementioned trend component. Again, the original trend (blue with triangles) and three modified trends (with factors $\gamma = 2.00$, $\gamma = 0.75$, $\gamma = 0.50$) are shown. All time series start at the same level and they keep the same trend changes, but their directions are different.

Season Determination Factor Let there be a factor κ that sets the season strength:

$$\underline{seas}^* = \kappa \cdot \underline{seas} \tag{5.6}$$

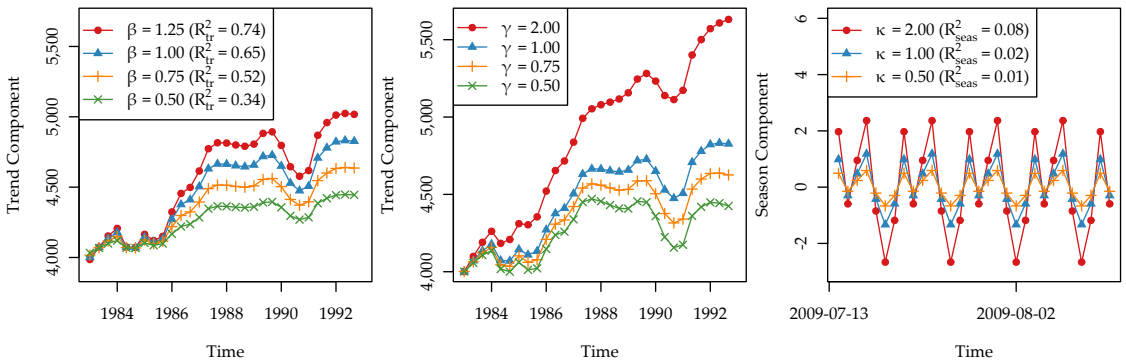
Depending on κ, R^2_{seas} increases ($\kappa > 1$), decreases ($0 \leq \kappa < 1$), or is left unchanged ($\kappa = 1$). $\kappa < 0$ is not admissible.

This effect is represented by Figure 5.3c on a season component from the Metering dataset. The plot shows the original season (blue with triangles) and two modified season components from the Smart Metering Project. The latter ones are modified by a season strength factor $\kappa = 2.00$ and $\kappa = 0.50$, respectively. Modifying the season by factor κ leads to higher peaks and lows. The resulting R^2_{seas} is given in the legend.

Feature Target Calculation

Based on time series features and factors, we are able to generate time series that cover given target features. Generated time series keep the nature of given time series except for the modified features.

Calculating the factor that corresponds to the target requires the calculation of the inverse function of a feature. We exemplify this by showing how a factor β is calculated that modifies the trend strength. Let $\underline{y}$ be a time series that shall be shifted to a target R^2_{tr} such



(a) Trend Strength (b) Trend Slope (c) Season Strength

Figure 5.3: Modification of Time Series Components

that its modified trend is $\underline{tr}^*$. Let $\underline{\tilde{tr}} = \theta_2 \cdot (\underline{t} - 1) + \underline{\delta_t}$ be the trend component without base value. Then:

$$\begin{aligned} R^2_{tr} &= 1 - \frac{var(\underline{res})}{var(\underline{res} + \underline{tr}^*)} \\ &= 1 - \frac{var(\underline{res})}{var(\underline{res} + \theta_1 + \beta \cdot \underline{\tilde{tr}})} \\ &= 1 - \frac{var(\underline{res})}{var(\underline{res}) + \beta^2 \cdot var(\underline{\tilde{tr}}) + 2 \cdot \beta \cdot cov(\underline{res}, \underline{\tilde{tr}})} \end{aligned} \tag{5.7}$$

where the covariance $cov(\underline{y}, \underline{y}')$ is defined as:

$$cov(\underline{y}, \underline{y}') = \frac{1}{T} \sum_{t=1}^{T} (y_t - m(\underline{y}))(y'_t - m(\underline{y}')) \tag{5.8}$$

The base value does not affect the variance and can be removed (Equation 5.7). Solving the following equation returns factor β, which is the positive real solution of the following equation:

$$\begin{aligned} 0 = {} & \beta^2 \cdot var(\underline{\tilde{tr}}) \cdot (R^2_{tr} - 1) \\ & + 2 \cdot \beta \cdot cov(\underline{res}, \underline{\tilde{tr}}) \cdot (R^2_{tr} - 1) \\ & + R^2_{tr} \cdot var(\underline{res}) \end{aligned} \tag{5.9}$$

This enables us to modify a time series such that its features are the target feature. For the other features this calculation is similar.

5.2.2 Feature-based Recombination

The feature-based generation technique described here is based on recombination and statistical models. Regarding these techniques, it is similar to Iftikhar et al. [ILD+17], but it extends several concepts in order to have a higher expressiveness. The technique generates time series that adhere to the feature-based representation. As shown in Figure 5.2, each time series from a given dataset is decomposed and characterized by its features. For a target $\underline{y}^*$, a time series is evolved as follows. First, the generation technique recombines deterministic components from a time series whose features fulfill the target feature. Thus, it selects a base value from a time series that satisfies the target, it selects a

trend slope from another time series that satisfies the target, and it selects season masks from different time series that satisfy the target. Second, it generates a stochastic component that satisfies the target and combines them with the deterministic components.

By taking *thresholds* into account, it allows the generation of time series with an expected distance to the target. Based on the feature-based distance, we define a *lower* and an *upper* threshold p and q. The time series $\underline{y}, \underline{y}' \in Y$ are similar with respect to a feature f_k if the feature-based distance d_k is within the thresholds:

$$p \leq d_k(\underline{y}, \underline{y}') \leq q \tag{5.10}$$

With q, we configure the maximum distance that is considered similar. This is an important notion for time series generation. While generation techniques from the literature do not define an upper threshold on standard distance measures, they can now be assessed by the feature-based distance. Likewise, the lower threshold p defines the minimum distance that two time series shall have. By default, this value is 0, thus ensuring perfect similarity. When this value is increased, a minimum deviation between two time series is expected which ensures that a generated time series is not fully equivalent to a given time series.

Recombination of Deterministic Components

The deterministic part of a time series $\underline{det}^\intercal = (det_1, det_2, ..., det_T)$ can be approximately recombined by its features as follows:

$$det_t = \theta_1 + (t-1) \cdot \theta_2 + \sum\nolimits_{s=1}^{S} \sigma_{s,(t-1)\%L_s+1} \tag{5.11}$$

where $\%$ is the modulo operator.

We use this relationship in order to draw *recombination candidates*. These candidates are time series that are similar to a given time series regarding their deterministic features. Thus, their recombination is also similar to the given time series.

The recombination comprises (1) the construction of a *distance matrix* for each deterministic feature, i.e., base value, trend slope, and season mask; (2) the *nearest neighbor search* to identify recombination candidates; and (3) the recombination based on features.

For every deterministic feature f_k, we calculate the distance $d_k(\underline{y}, \underline{y}')$ between every pair of given time series. These distances are stored in a distance matrix.

Subsequently, a nearest neighbor search is carried out on the distance matrix to identify neighboring features that fulfill the error thresholds p and q. For every deterministic feature f_k, each given time series (identified by i) is annotated with a set of candidates (identified by j) that fulfill the condition $p \leq d_k(\underline{y}_i, \underline{y}_j) \leq q$.

Finally, we carry out the recombination. From the recombination candidates, we randomly select one candidate per feature, i.e., one candidate j_{θ_1} for the base value, one candidate j_{θ_2} for the trend slope and $\sum_s L_s$ candidates $j_{\sigma_{s,l}}$ for the season masks. Their recombination yields the deterministic part $\underline{det}_i^*$ of the generated time series $\underline{y}_i^*$:

$$det_{i,t}^* = \theta_1(\underline{y}_{j_{\theta_1}}) + (t-1) \cdot \theta_2(\underline{y}_{j_{\theta_2}}) + \sum\nolimits_{s=1}^{S} \sigma_{s,(t-1)\%L_s+1}(\underline{y}_{j_{\sigma_{s,(t-1)\%L_s+1}}}) \tag{5.12}$$

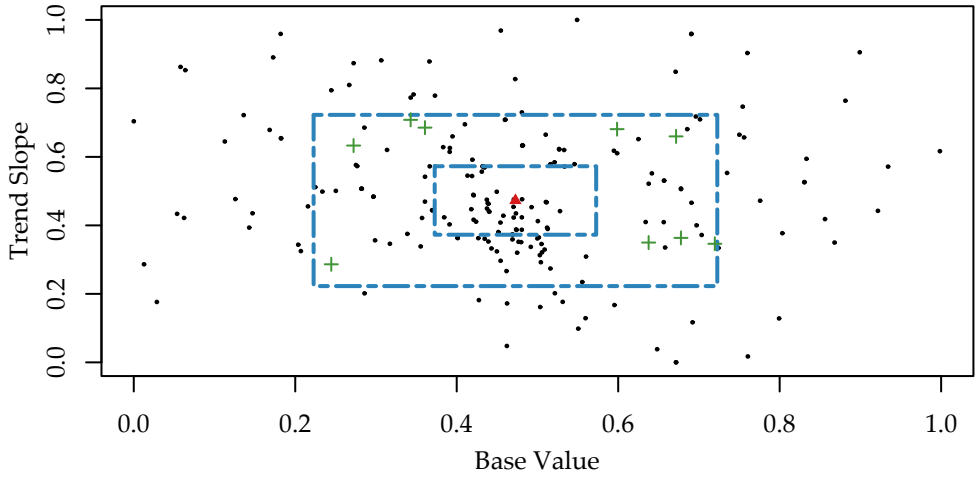


Figure 5.4: Feature Space of Economy Dataset

Figure 5.4 illustrates the feature space of the Economy dataset for two 0-1-normalized features with outliers, the base value and trend slope. We assume a lower error threshold $p = 0.10$ and an upper error threshold $q = 0.25$. For an example time series (red triangle in the center), possible recombination candidates lie within the error thresholds (black points between the inner and the outer blue dashed rectangle). They are recombined and result in new combinations that still fulfill the feature-based similarity (green crosses). These recombinations are the deterministic part of generated time series.

There are two anomalies that can occur. If two recombinations are based on the same features, they are treated as duplicates. If a recombination matches the components of a given time series, it is treated as the original time series. Both anomalies are analyzed in order to make sure that they do not influence the results of the generation.

Simulation of Stochastic Component

The stochastic component includes the remaining random information that cannot be modeled directly. In order to describe this component, its value distribution and its remaining autocorrelation can be used. Our feature-based representation captures these characteristics with the features: standard deviation, skewness, kurtosis, and autocorrelation of lag 1 (Equations 3.40, 3.43, 3.44, and 3.45, page 36f.).

To generate a component that agrees with these features, we apply a *composite* statistical model. It combines a distribution function generating random values with a simulation of an autoregressive model. Moreover, we take care that the generated residuals do not exceed the value limits from the given dataset by imposing constraints to the generated residuals. These three elements are explained subsequently.

Distribution Function The goal is to generate residuals that agree with the target distribution features. A distribution that fits this purpose is the *Pearson distribution system*. It is a set of eight different distribution functions that cover a large space of distribution features [BK17].

For each feature combination $\{sd(\underline{y}), skew(\underline{y}), kurt(\underline{y})\}$, a distribution function is selected based on its ability to provide a sample for this combination. The mean is assumed to be

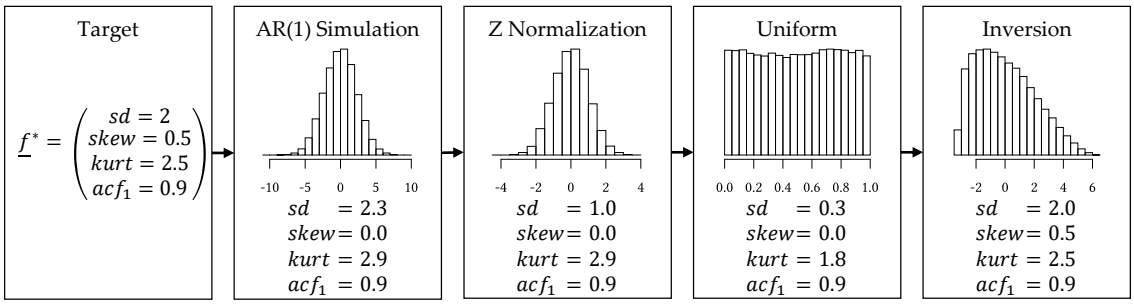


Figure 5.5: Autoregressive Model

0 since it is captured by the base value. For a given time series $\underline{y}$ and error thresholds p and q, a time series is generated whose distribution features $\underline{f}$ are expected to be in the interval below the given features, $[f^{01out}(\underline{y}) - q, f^{01out}(\underline{y}) - p]$, or the interval above the given features, $[f^{01out}(\underline{y}) + p, f^{01out}(\underline{y}) + q]$, where f^{01out} is the normalized feature of standard deviation, skewness, and kurtosis.

Autoregressive Model In the presence of significant autocorrelation, an autoregressive model is simulated. It weaves autocorrelation into the generated residuals such that the residuals approximately agree with both, the distribution and and the autocorrelation features. Figure 5.5 illustrates this weaving, starting with expected *target* features. We first *simulate* an autoregressive process using an $AR(1)$ model. As presented in Section 5.1, an $AR(1)$ model is of the form:

$$y_t = \phi_1 \cdot y_{t-1} + a_t \tag{5.13}$$

The autocorrelation of lag 1 equals ϕ_1. Thus, we simulate the process for the given target feature $acf_1(\underline{res}) = \phi_1$, leading to values with the expected autocorrelation. However, the simulated values are normally distributed, since the error terms a_t are normally distributed. To support other distribution features, we transform the values to the target distribution by means of inversion sampling. The simulation from the autoregressive model is z-normalized, with a mean of 0 and a standard deviation of 1, which keeps the autocorrelation from the simulation. Then, it is transformed to a *uniformly distributed* random variable U using $u = F_N(y)$, where F_N is the cumulative distribution function of the normal distribution $N(0, 1)$. The values of U are between 0 and 1, and their autocorrelation of lag 1 is still as expected. Finally, by using the inverse of the cumulative distribution function $F_{Pearson}$ from the Pearson distribution system, the *inversion sampling* yields residuals $res^* = F^{-1}_{Pearson}(u)$ that reproduce the expected distribution and autocorrelation features.

Constraints The simulation of the aforementioned model provides residuals that agree with the distribution and autocorrelation features of a given time series. However, once the residuals are combined with the deterministic components they may lead to unexpectedly low or high time series values. For example, such combinations may lead to a negative or extremely high wind speed value which should be avoided.

Therefore, we define two *constraints* min_Y and max_Y that express the lowest possible and the highest possible value in a dataset, respectively. The generation of residuals is aware of these constraints and of the deterministic part of the time series they were combined with. It ensures that evolved residuals do not exceed these constraints as follows.

Table 5.2: Properties of Proposed Generation Techniques

	MD	**FBG**
Dataset-oriented	–	✓
Deterministic	✓	✓
Stochastic	–	✓
Innovative	✓	✓

Feature-based Modification (MD);
Feature-based Recombination (FBG)

An error term a_t, which is a set of samples from a normal distribution, is generated. At time instance t, a new value y_t^* from the autoregressive model is simulated using a_t. If the resulting residual value res_t^* does not fulfill the conditions:

$$min_X - det_t^* \leq res_t^* \text{ and } res_t^* \leq max_X - det_t^* \tag{5.14}$$

it is rejected and another sample for a_t is drawn. The originally selected value a_t is used at another future time instance. Thus, it ensures the distribution properties of the sample.

5.2.3 Comparison

Table 5.2 summarizes the properties of the proposed generation techniques. The generation technique based on modification (MD) is deterministic and innovative. The use of factors allows to modify the influence of a deterministic component such that an evolved time series fulfills target features. Moreover, it incorporates new characteristics in a generated dataset that were not provided by a given dataset. While it is a valuable technique to create "what-if" scenarios of a dataset it is not dataset oriented and it does modify the stochastic properties. Moreover, the trend changes $\underline{\delta}$ are taken into account which is why this approach does not fully reduce the representation of a time series.

The generation technique based on generation (FBG) fulfills all properties: (1) by recombining time series, it is dataset-oriented, (2) by reusing trend and season components, it is deterministic, (3) by simulating residuals with expected distribution and correlation characteristics, it is stochastic, and (4) by relying on modifiable features, it is innovative. As stated in the introduction of this chapter, our main objective is the applicability of the generation technique across domains. Subsequently, we assess FBG to generation techniques regarding three domains, and compare it with other generation techniques.

5.3 EXPERIMENTAL EVALUATION

The feature-based generation technique FBG provides desirable properties as it reproduces deterministic and stochastic characteristics. In this section, we evaluate that (1) FBG evolves a dataset that is highly similar to a given dataset regarding the feature-based distance, and that (2) the feature-based distance measure is able to compare generation techniques regarding their expressiveness.

We evaluate our technique with the three time series datasets Metering, Wind, and Economy that have been introduced in Section 4.1 (page 42). The Metering and Economy datasets have already been used by Iftikhar et al. and Kang et al., respectively [ILD+17, KHSM17]. In order to extract the yearly season of the Metering dataset (which needs at least two cycles), it is assumed that the months January 2011 until June 2011 behave like January 2010 until June 2010. The Wind dataset is similar to the dataset used by Pesch et al. [PSAH15].

5.3.1 Experimental Setting

We compare FBG to three generation techniques that have been recently cited in the literature (Table 5.3). Each technique is applied to a dataset from a specific domain and is used to reproduce either all deterministic and stochastic features or a subset of them. For each dataset Y, a generation technique evolves one dataset Y^*, i.e., for each time series $\underline{y_i} \in Y$, it evolves one time series $\underline{y_i^*} \in Y^*$. Thus, the size of the generated dataset is equal to the size of the given dataset with one exception: only 100 time series are generated for the Metering dataset which is due to time limitations of the genetic algorithm. The comparison is carried out regarding standard and feature-based distance measures. To ensure the reproducibility of the results, we summarize the configuration of the selected techniques. Our technique, FBG, is set to a lower error threshold $p = 0.01$ and an upper threshold $q = 0.05$ for all features and all datasets.

The generation technique from Iftikhar et al. [ILD+17] is based on *recombination*. It splits a time series into a base component, a season mask, and residuals. The base component is the long-term mean value of the time series, without a trend. The season masks are then clustered, shuffled, and assigned to another base component and residuals from the same cluster. We carry out a k-means clustering of the daily season masks with the Euclidean distance measure. The number of clusters is 20 as proposed by Iftikhar et al. We apply this setting to the Metering and the Economy dataset. We create only 5 clusters for the Wind dataset because it contains less time series. The residuals are simulated with an AR(3) model as suggested by Iftikhar et al. Since a trend is not considered by this technique, we do not assess the trend slope with the feature-based distance. For assessing the raw-data-based distance, we add the original trend.

A second comparison is carried out for *Markov chains*. The generation technique of Pesch et al. [PSAH15] is re-implemented with some slight modifications. Each time series is decomposed into deterministic and stochastic components. A Markov chain of second order with 65 states (Metering, Wind) and a second order lag of 6 capture the state transitions of a given time series. These parameters correspond to the suggestion of Pesch et al. We only set 10 states for the Economy dataset because its time series are too short to fill a transition probability matrix for 65 states. In the rare case of anomalies in the transition probability matrix, a healing mechanism is applied as suggested by Pesch et al. In contrast, we adopt the multi-seasonal decomposition instead of the trend and season elimination that the authors applied. Moreover, we do not apply a running average filter for smoothing the time series because this characteristic is not important for this work. Generation is carried out for the stochastic component only since Markov chains are not applicable to deterministic components.

Finally, we re-implement the *genetic algorithm* based on Kang et al. [KHSM17]. As suggested by the authors, the initial population consists of 20 time series Economy that are randomly selected. Its size is set to 20 for the Metering and 10 for the Wind dataset. The time series $\underline{y_i}$, which sets the target for the generation of $\underline{y_i^*}$, is not among the initial population. A crossover of time series occurs with a probability of 80%, a mutation occurs with a probability of 40%. Time series are selected by their fitness. We use the overall feature-based distance d_f as the fitness function (Equation 4.9, page 50). As suggested by the authors, the iteration stops if (1) the fitness is at least -0.01, (2) if the maximum number of iterations, 3000, has been reached, or (3) if there was no improvement in a sequence of 200 iterations.

Table 5.3: Experimental Setting

Generation Technique	Dataset	Relevant Features
Recombination	Metering	All without trend slope
Markov chain	Wind Speed	Stochastic
Genetic algorithm	Economy	All
FBG	All	All

5.3.2 Feature-based Distance

The selected generation techniques are compared regarding the feature-based and the overall feature-based distance (Equations 4.8 and 4.9, page 50). First, they are applied to the domain they were originally designed for, second they are applied to all domains.

Recombination on Metering

Figure 5.6 compares FBG with the recombination technique. The x-axis shows the feature f_k, the y-axis shows the distances $\{d_k(\underline{y}_i, \underline{y}_i^*), 1 \leq i \leq I\}$ as boxplot. For readability reasons, seasonal features of the same seasonal component are presented in one box.

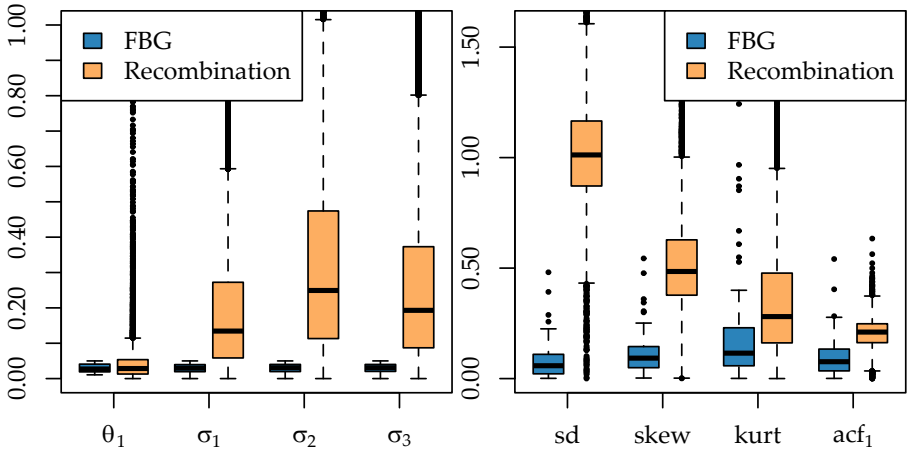
FBG generates time series whose deterministic features respect the error threshold q (Figure 5.6a). The base value also respects the error threshold p. However, there are rare cases where the seasonal features of a given time series is not recombined which led to a feature-based distance below p. The recombination technique generates time series with a higher feature-based distance to its given counterpart. While the base features (which are not recombined) yield good results, the clustered and recombined daily season masks are more distant.

FBG also generates stochastic features which are similar to their given counterparts (Figure 5.6b). Although the distribution features from FBG are remarkably good, they are not below the expected error threshold q. The simulation process makes a trade-off between the expected features and the value constraints. The recombination technique uses an AR simulation that generates values with a fixed standard deviation, skewness, and kurtosis. Thus, these features do not fit well the given moments of the original dataset. The autocorrelation can be well reproduced.

Markov Chain on Wind

The Markov chain technique evolves only stochastic components. The generated dataset reproduces well the given features for the Wind dataset (Figure 5.7). FBG yields better results but the skewness is higher than for the Markov chain due to the trade-off with the value constraints.

The feature-based distances seem high because the Wind dataset is very homogeneous. However, the non-normalized distances are not (Figure 4.3, page 49).



(a) Deterministic Features (b) Stochastic Features

Figure 5.6: Feature-based Distance of Recombination

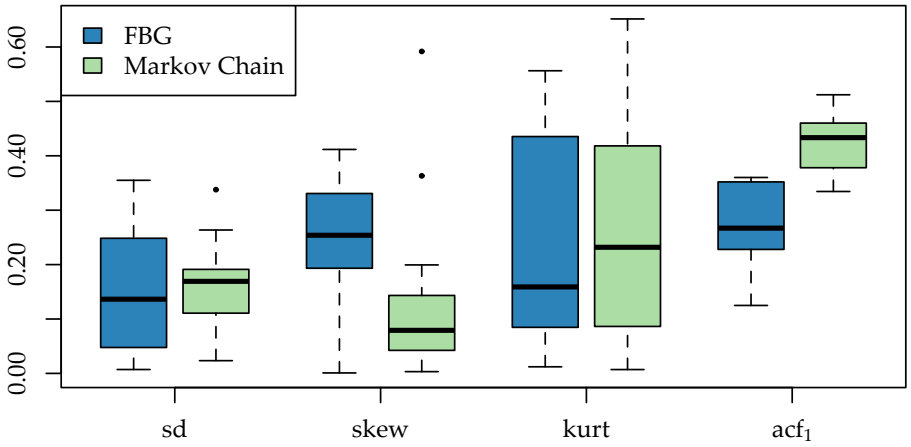
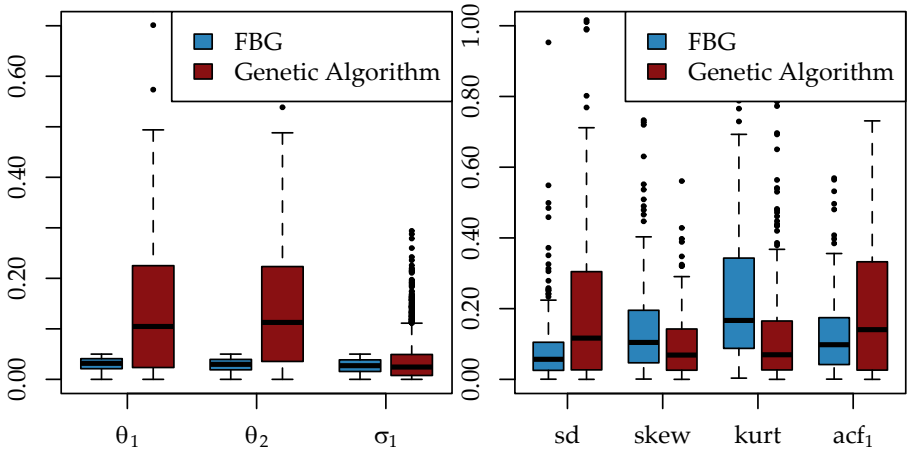


Figure 5.7: Feature-based Distance of Markov Chain



(a) Deterministic Features (b) Stochastic Features

Figure 5.8: Feature-based Distance of Genetic Algorithm

Genetic Algorithm on Economy

Figure 5.8 compares FBG with the genetic algorithm on the Economy dataset. For the deterministic features, FBG does not exceed the error threshold q (Figure 5.8a). In less than 8%, no recombination candidates are found which is why the component is identical to the given one. Thus, the lower whiskers reach 0 instead of stopping at p. The genetic algorithm generates highly similar components. The base value and trend slope have a median feature distance of 10% while the season masks differ from given season masks by only 2%.

Regarding the stochastic features, FBG provides more similar residuals in terms of standard deviation and autocorrelation (Figure 5.8b). However, the genetic algorithm outperforms it regarding skewness and kurtosis for the same reason as in the experiments above.

Summary

Table 5.4 lists the median feature-based distance per dataset and generation technique. For readability reasons, seasonal features of the same seasonal component are printed in the same cell. The table also lists the median overall feature-based distance d_f. It is calculated based on the relevant features (Table 5.3), other features (cell crossed-out) are ignored. The best distances are printed in boldface.

FBG generally outperforms the other techniques on the deterministic components due to the error thresholds. It partially yields the best results on the stochastic components. However, Markov chains are sometimes better because they use a bigger model for generating residuals. The good results of the genetic algorithm on the Economy dataset are due to the 3000 iterations which led to best results on some of these features.

FBG also outperforms the other techniques on the overall feature-based distance. The genetic algorithm is also very accurate on the Economy dataset. The Markov chain technique performs well on the Metering dataset, however, it does not take the deterministic components into account.

5.3.3 Standard Distance

The feature-based distance measure compares characteristics that generation techniques are able to express. In this subsection, we show that it is related to the results from standard distance measures that we presented in Section 5.1: the *raw-data-based* distance, the *distribution* distance, and the *autocorrelation* distance. We assess them numerically as well as visually (Figure 5.9).

Raw-data-based Distance

Visually, the raw-data-based distance is assessed with a lineplot. Figure 5.9a shows the first four years of an exemplary time series from the Economy dataset (black solid line) together with the result from FBG (blue dashed line) and from the genetic algorithm (red dotted line). Due to the error thresholds and the strong deterministic components, the

Table 5.4: Median Feature-based Distance on All Datasets

	Deterministic Features					Stochastic Features				Overall
Metering	θ_1	θ_2	σ_1	σ_2	σ_3	**sd**	**skew**	**kurt**	$\mathbf{acf_1}$	$\mathbf{d_f}$
RE	0.0285	–	0.1344	0.2492	0.1930	1.0120	0.4846	0.2805	0.2105	0.2935
MC	–	–	–	–	–	**0.0388**	**0.0494**	**0.0691**	**0.0367**	0.0587
GA	0.1145	–	0.0977	0.0880	0.1244	0.1025	0.2063	0.3159	0.1970	0.1613
FBG	**0.0272**	–	**0.0296**	**0.0306**	**0.0306**	0.0582	0.0926	0.1151	0.0764	**0.0410**
Wind	θ_1	θ_2	σ_1	σ_2	σ_3	**sd**	**skew**	**kurt**	$\mathbf{acf_1}$	$\mathbf{d_f}$
RE	0.0494	–	0.2703	0.5889	–	1.3999	2.2606	0.9540	1.9901	0.6434
MC	–	–	–	–	–	0.1692	**0.0792**	0.2318	0.4334	0.2636
GA	0.2078	0.2520	0.0749	0.1881	–	0.2469	0.1850	0.3778	0.6589	0.2137
FBG	**0.0192**	**0.0190**	**0.0180**	**0.0184**	–	**0.1364**	0.2539	**0.1590**	**0.2670**	**0.0653**
Economy	θ_1	θ_2	σ_1	σ_2	σ_3	**sd**	**skew**	**kurt**	$\mathbf{acf_1}$	$\mathbf{d_f}$
RE	–	–	0.2193	–	–	0.3517	0.1429	0.3325	0.2440	0.3867
MC	–	–	–	–	–	0.0575	0.1245	0.1896	0.2663	0.2222
GA	0.1047	0.1127	**0.0244**	–	–	0.1166	**0.0687**	**0.0696**	0.1406	0.0997
FBG	**0.0299**	**0.0294**	0.0275	–	–	**0.0349**	0.1199	0.1548	**0.0886**	**0.0767**

FBG result is very close to the original time series. The genetic algorithm has some peaks and lows due to mutation.

Numerically, the raw-data-based distance is calculated as RMSE. Table 5.5 lists the median RMSE of a given and the corresponding generated time series per dataset and generation technique. On the Metering dataset, FBG is the most accurate regarding raw-data-based distance. The Markov chain technique ranks second. Although this technique copies the deterministic components as is, it has a higher RMSE. One reason is that it does not take value constraints into account. On the Wind dataset, the genetic algorithm yields the highest accuracy. Due to the homogeneous data, every time series in the initial population is already highly similar to the time series that set the target features. The techniques Markov chain and FBG return a similar distance. While the Markov chain technique copies the deterministic components, FBG recombines components with a lower error threshold that lead to a slightly higher RMSE. On the Economy dataset, the recombination technique shows the lowest distance. Since this technique does not exhibit a high similarity of monthly season masks (σ_1), the trend component that is copied as is has a major influence on this accuracy.

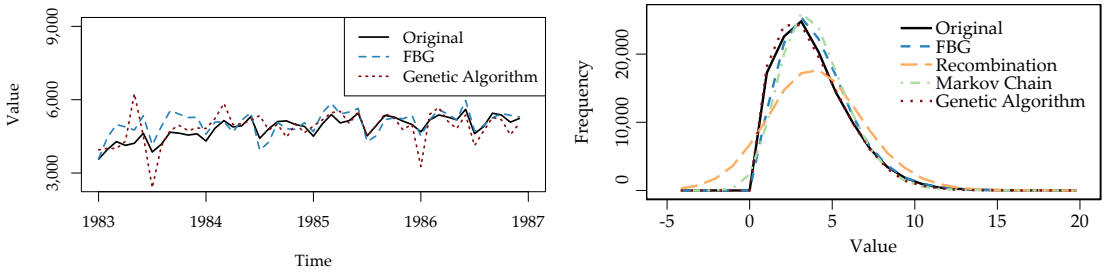
Distribution Distance

The distribution distance focuses on *how often* a time series value occurs rather than *when* it occurs. Visually, Figure 5.9b illustrates the histogram of an original Wind time series and the corresponding generated ones. The genetic algorithm, FBG, and the Markov chain generate values that are highly similar to the original ones. The recombination technique is less similar because it does not take into account the higher order moments.

Numerically, we assess the distribution distance with a histogram distance measure, the *Bhattacharyya distance measure* [Bha43]:

$$d_B(\underline{y}, \underline{y}^*) = -\log \sum_{i=1}^{A} \sqrt{freq_i(\underline{y}) \cdot freq_i(\underline{y}^*)} \tag{5.15}$$

where i is one of A equidistant buckets and $freq_i$ is the relative frequency of values within the bucket. Equal distributions have a distance 0, the less similar they are the higher is



(a) Lineplot of FBG and Genetic Algorithm (b) Histogram of Residuals on Wind Dataset

Figure 5.9: Visual Distances

the distance. Table 5.5 lists the median Bhattacharyya distance for every dataset and generation technique. We split the value range into $B = 100$ (Metering), $B = 30$ (Wind), and $B = 10$ (Economy), respectively. On the Metering and Wind datasets, the genetic algorithm ranks first and FBG second. The Metering dataset contains several deterministic effects that are not captured by our time series model. Thus, they are treated as residuals and they are more difficult to reproduce. The Wind dataset is very homogeneous. New combinations and mutations of its time series reproduce the value distribution very well. Therefore, the genetic algorithm ranks first, followed by FBG that reproduces the distribution from only three distribution features. The Markov chain and recombination technique yield less accurate distribution distances which is mainly due to missing value constraints as shown in Figure 5.9b. On the Economy dataset however, the Markov chain and recombination technique yield better results than the competitors. Since these techniques take the strong trend component as is, they could better reproduce the original distribution than FBG and the genetic algorithm. Moreover, these time series are very short and thus, FBG is unable to reproduce the value distribution accurately.

Autocorrelation Distance

We assess the non-normalized autocorrelation of lag 1 of original and generated time series (Equation 3.45, page 37). Table 5.5 lists the median distance per dataset and technique. These results correspond to the normalized feature acf_1. FBG has the best results on the Wind and Economy dataset. The Markov chain has the smallest distances on the Metering dataset. On the Economy dataset, its results are only slightly worse than those of FBG.

Summary

The standard distance measures suggest that FBG competes with generation techniques from the literature. Regarding the distribution and ACF distance, it mostly ranks first and second place. Regarding the raw-data-based distance, it ranks first place on Metering, but third place on Wind and Economy. As mentioned earlier, two competitors have an advantage as they take the given trend (RE) and the given deterministic components (MC) as is. The feature-based and overall feature-based distances correspond to standard distance measures only to some extent. They compare the similarity in structure, while the raw-based-distance measure compares the similarity in shape. The distribution features do not capture the full spectrum of a histogram which is why these features only correspond to the histogram - and thus, the histogram distance - in special cases.

Table 5.5: Standard Distances

Metering	**Raw-data-based**	**Distribution**	**ACF**
RE	1.7829	0.4471	0.2320
MC	0.7934	0.1478	**0.0183**
GA	0.8214	**0.0858**	0.1562
FBG	**0.7167**	0.1305	0.0492
Wind	**Raw-data-based**	**Distribution**	**ACF**
RE	3.4251	0.0676	0.1142
MC	2.7296	0.0242	0.0205
GA	**2.0618**	**0.0090**	0.0336
FBG	2.7559	0.0149	**0.0166**
Economy	**Raw-data-based**	**Distribution**	**ACF**
RE	**277.4685**	0.0790	0.0268
MC	335.3713	**0.0767**	0.0146
GA	690.3626	0.2520	0.0200
FBG	387.8705	0.0999	**0.0145**

5.4 SUMMARY

In this chapter, we introduced feature-based time series generation. We reviewed and analyzed generation techniques from the literature and we derived properties they have to fulfill to be applicable across domains. Moreover, we reviewed distance measures that assess the expressiveness of generation techniques. We hypothesized that our feature-based representation captures these characteristics well.

Based on these reviews, we introduced feature-based generation techniques. The generation technique based on recombination, FBG, supports the desired properties and evolves highly similar time series. Moreover, it respects similarity expectations which are expressed by the feature-based distance and limiting error thresholds. By this means, it outperforms currently available generation techniques and confirms our hypothesis.

This evaluation is the first to present a domain-independent comparison of time series generation. This assessment is important in order to generalize generation techniques from different domains that were considered isolated applications but that could evolve datasets independent of their original application. In this context, the feature-based distance is a tool for the validation of generated datasets. Generated time series are selected if they respect an expected feature range which is required by an application.

6

TIME SERIES MATCHING

EXHAUSTIVE TIME SERIES GATHERING and the inherent nature of time series lead to the design of data management systems handling this data type natively [ZP18]. These systems provide a storage model, a query language, and optimization mechanisms suitable for time series. Fast data access is needed to carry out data-mining tasks efficiently. A crucial prerequisite of these systems is the retrieval of similar time series which is commonly referred to as time series matching [ZP18]. Figure 6.1 gives an overview of this data-mining task. Due to their high dimensionality, a *query time series* is not directly matched against a *time series dataset* to retrieve the *match*, i.e., its nearest neighbor. Instead, both objects are *represented* and compared in a low-dimensional space. Their representation distance approximates their true distance which is assumed to be the Euclidean distance (Equation 3.2, page 24) [AFS93]. Two *matching methods* are carried out on these objects: *exact* and *approximate* matching. Exact matching retrieves the *exact* match which is closest to the query time series with respect to the Euclidean distance. In contrast, approximate matching retrieves the *approximate* match which is closest to the query time series with respect to the representation distance.

For the past three decades, researchers have been developing engineering techniques for matching that rely on the shape [DTS+08], on the model parameters [KGP01], or on the features [AFS93] of a time series. Among these techniques, the symbolic aggregate approximation (SAX) from Lin et al. is of particular interest [LKLC03], as presented earlier (Section 3.2, page 26). First, this shape-based technique splits a time series into segments which are represented by their mean value, the so-called piecewise aggregate approximation (PAA). Second, it discretizes each mean value by mapping it to a discrete symbol. Thus, SAX provides a small representation together with a fast distance measure which makes it suitable for time series matching. Moreover, its distance measure has the important property to lower-bound the Euclidean distance measure, i.e., it allows for pruning time series observations based on the representation, without the need to load all high-dimensional time series into memory and calculate their Euclidean distance.

However, this technique suffers from two design criteria. First, SAX assumes that the PAA of a z-normalized time series, whose values have a mean of zero and a variance of one, is also normally distributed with the same variance. As pointed out by Butler and Kazakov, this is over-simplistic and negatively impacts the symbolic distribution [BK15]. Second, SAX ignores the deterministic behavior of a time series. As shown in our design rationale (Section 4.1, page 42), a season and a trend are common behavior in many domains. SAX does not take it into account which again leads to a distortion in the symbolic distribution.

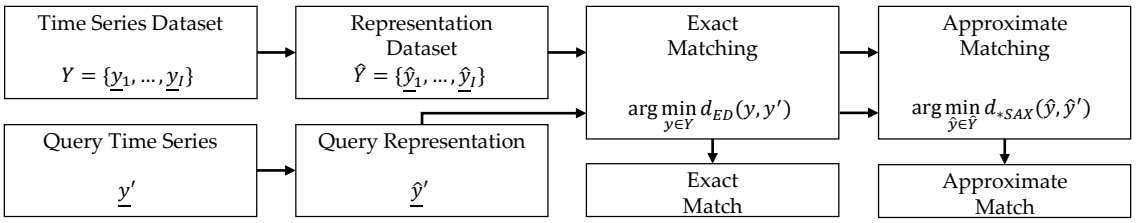


Figure 6.1: Overview of Time Series Matching

This chapter focuses on engineering techniques based on SAX that aim to solve these two shortcomings and thus, provide a more efficient time series matching.

Several SAX extensions have been proposed to include further characteristics. While the original SAX focuses on the mean value of a segment, existing SAX extensions also include its standard deviation or its trend. To the best of our knowledge, no work in the literature focuses on global features before the time series is split into segments. As we will show, features from deterministic components leverage SAX for a more efficient matching while providing a lower-bounding distance measure.

In this chapter, we introduce the season- and trend-aware symbolic approximations, *sSAX* and *tSAX*, that take into account the global time series' season and trend, respectively. First, they improve the symbolic distribution of the representation compared to SAX. Second, they provide a more accurate representation while keeping the same representation size as SAX. Taking into account this behavior enables them to provide a more accurate and efficient time series matching.

In this chapter, we shortly recap SAX and provide a comprehensive review of existing SAX extensions. Although all extensions provide lower-bounding distance measures, most of them increase the representation size (Section 6.1). We introduce our techniques sSAX and tSAX which provide a high accuracy at the same representation size as SAX. As we will show, their distance measures are also lower-bounding, which is most important (Section 6.2). Subsequently, we evaluate the techniques for time series matching (Section 6.3) where we summarize our experimental setting and discuss our results. The most remarkable result to emerge from this evaluation is that on big datasets (100 Gb), sSAX returns exact matches up to three orders of magnitude faster than SAX. We conclude with a summary of our observations (Section 6.4). Throughout this chapter, we use the term "normalization" to refer to "z-normalization" because it is the only normalization technique assumed by Lin et al. [LKLC03].

6.1 STATE OF THE ART

Many attempts have been made to include further features in SAX representations. In this section, we give an overview of these attempts. We start by giving a short recap of the original SAX as presented in Section 3.2 (page 26) and a discussion of its properties for an efficient time series matching. Subsequently, we review SAX extensions and assess them regarding these properties.

6.1.1 Original SAX

The original SAX reduces the dimensionality of a time series in two steps: the segment-wise aggregation into mean values using PAA and the discretization into symbols using SAX. The former reduces the dimensionality in the time domain, while the latter reduces the dimensionality in the value domain. As such, PAA is the prerequisite of SAX as it aggregates a time series as follows:

$$\underline{\bar{y}}^{\mathsf{T}} = (\bar{y}_1, ..., \bar{y}_w, ..., \bar{y}_W) \tag{6.1}$$

where

$$\bar{y}_w = \frac{W}{T} \sum_{t=\frac{T}{W}(w-1)+1}^{\frac{T}{W}w} y_t \tag{6.2}$$

Still, this representation contains real values that take a considerable amount of storage, which is why it is further discretized utilizing SAX:

$$\underline{\hat{y}}^{\mathsf{T}} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_w, ..., \hat{y}_W) \tag{6.3}$$

where

$$\hat{y}_w = \begin{cases} 1 & -\infty < \bar{y}_w < b_1 \\ a & \exists a : b_{a-1} \leq \bar{y}_w < b_a \\ A & b_{A-1} \leq \bar{y}_w < \infty \end{cases} \tag{6.4}$$

As such, $\hat{y}$ is the mean value $\bar{y}$ discretized into an alphabet $A \in \mathbb{N}_{>0}$, and $\underline{b}^{\mathsf{T}} = (b_1, ..., b_a, ..., b_{A-1})$ are the breakpoints of this alphabet:

$$]-\infty, b_1[, ..., [b_{a-1}, b_a[, ..., [b_{A-1}, \infty[\tag{6.5}$$

SAX reduces each mean value to a discrete symbol of the alphabet A. Ideally, the symbols of a dataset are equiprobable so that they make full use of the alphabet capacity. For achieving this, Lin et al. assume that mean values would be $\mathcal{N}(0,1)$-distributed because the time series are normalized [LKLC03]. Consequently, breakpoints are set such that the area under the normal distribution $\mathcal{N}(0,1)$ from $[b_{a-1}, b_a[$ equals $1/A$.

The SAX representation of a time series has a size $W \cdot ld(A)\ bit$, which is small compared to the original time series occupying $T \cdot 32\ bit$. Moreover, SAX allows for a fast transformation requiring only one pass over the dataset. The time series are assumed to be normalized beforehand in a preprocessing step.

Distance measures for PAA and SAX are defined as follows:

$$d_{PAA}(\underline{\bar{y}}, \underline{\bar{y}}') = \sqrt{T/W}\sqrt{\sum_{w=1}^{W}(\bar{y}_w - \bar{y}'_w)^2} \tag{6.6}$$

$$d_{SAX}(\underline{\hat{y}}, \underline{\hat{y}}') = \sqrt{T/W}\sqrt{\sum_{w=1}^{W} cell(\hat{y}_w, \hat{y}'_w)^2} \tag{6.7}$$

where

$$cell(a, a') = \begin{cases} 0 & |a - a'| \leq 1 \\ b_{max(a,a')} - b_{min(a,a')+1} & otherwise \end{cases} \tag{6.8}$$

The results of $cell(a, a')$ are stored as a lookup table occupying $A^2 \cdot 32\ bit$, which is rather small for typical alphabet sizes and which is calculated once per dataset. Thus, the distance calculation requires one lookup per segment, so in total W lookups for comparing two time series. This is faster than the Euclidean distance where the calculation consists in loading two time series of length $T \gg W$ in memory and comparing them value by value. Moreover, the SAX distance measure lower-bounds the Euclidean distance, which has been proven for its prerequisite PAA in [YF00] and for SAX itself in [LKWL07].

An engineering technique that competes with SAX should provide similar properties. Moreover, by including other features it should also give a higher representation accuracy and a more efficient time series matching. These properties, however, must be evaluated experimentally.

6.1.2 SAX Extensions

Several SAX extensions that include further features have been proposed in the literature. We review them regarding their representation and distance properties and summarize them in Table 6.1 together with the original SAX.

ESAX [LSK06] extends SAX by taking the extreme values of each segment into account. Thus, it represents a PAA segment by a minimum, mean, and maximum symbol, which are retrieved simultaneously. Although this technique is more accurate than SAX, it triples the representation size. For each segment, the distance is calculated with the help of a lookup table. However, the shape of the lookup table is not given. Since there are three symbols per segment, a lookup table would have a size A^6. Therefore, we assume that ESAX cannot fully precalculate the distances.

1d-SAX [MGQT13] segments a time series first using piecewise linear approximation (PLA) and discretizes the values using SAX. PLA represents each segment by its mean level and its slope which are estimated by linear regression. Subsequently, these features are discretized like SAX and interleaved to one representation. Thus, they have the same representation size as SAX. For the representation, it needs a second pass over the data due to the linear regression. The distance calculation is based on a lookup table and requires W lookups. It is only formulated for an asymmetric comparison, i.e., the distance of the real-valued query and the discretized observations. Therefore, one lookup table is constructed per query that stores the distance between each segment of the query and each symbol of the alphabet.

TD-SAX [SLL+14] also uses segment trends which are encoded as start and end value of each segment. Although fast to calculate, these real values cause TD-SAX representations to be much larger than SAX. The distance calculation for the mean values relies on W lookups and thus, it needs the same storage as SAX. However, the trend distance is calculated on real values and does not use a lookup table. Therefore, the distance calculation has additional costs.

TFSA [YYZ+15] represents a time series by segment trends only. It splits the time series into segments of unequal length using a changepoint detection algorithm that passes twice over the time series. On every segment, trends are extracted by linear regression (third pass) and discretized in $2\ bit$: increasing, decreasing, or stationary. The slope and the endpoint of each trend annotate this symbol, but they are not discretized. Compared to SAX, this engineering technique has an increased representation size. TFSA calculates the distance from the representation without a lookup table. This practice might be slower because there are several floating-point operations involved for each segment.

Table 6.1: Properties of Engineering Techniques

Technique	Representation		Distance		
	Size (bit)	Time	Storage (32 bit)	Time	LB
SAX	$W \cdot \text{ld}(A)$	1	A^2	W	✓
ESAX	$3 \cdot W \cdot \text{ld}(A)$	1	A^6	W	✓
1d-SAX	$W \cdot \text{ld}(A)$	2	$W \cdot A$	W	(✓)
TD-SAX	$W \cdot (\text{ld}(A) + 32) + 32$	1	A^2	W	✓
TFSA	$W \cdot (\text{ld}(T) + 66)$	3	0	W	✓
SAX_SD	$W \cdot (\text{ld}(A) + 32)$	1	A^2	W	✓
sSAX	$W \cdot \text{ld}(A)$	1	$A_{seas}^2 + A_{res}^2$	$4WL$	✓
tSAX	$W \cdot \text{ld}(A)$	2	$A_{tr}^2 + A_{res}^2$	$W + 1$	✓

Representation Time in passes over time series; Distance time in number of lookups; LB means lower-bounding

SAX with standard deviation (SAX_SD) [ZY16] represents every PAA segment with its mean value (discretized as SAX) and its standard deviation (not discretized). Both features are calculated in one pass but they increase the representation size. The distance calculation is done with a lookup table for the mean value. The distance of the standard deviation is calculated directly on the feature.

Based on Table 6.1, we make the following observations:

Representation Size All SAX extensions increase the representation size except 1d-SAX. For an unbiased evaluation of representation accuracy, it should be equal.

Representation Time 1d-SAX and TFSA need several passes over the dataset for the representation, which is an acceptable penalty because the calculation still has linear complexity.

Distance Storage The distance storage often uses a lookup table with a small size that provides a fast distance calculation.

Distance Time The distance calculation needs W lookups for all techniques. However, for features other than the mean value, the distance calculation has additional costs. A detailed evaluation would require an optimized distance function of each technique which is not covered in this work.

Lower-bounding Distance The distance measures of ESAX, TD-SAX, TFSA, as well as SAX_SD lower-bound the Euclidean distance measure. Although PLA is lower-bounding [CCL+07], it is not clearly stated for 1d-SAX.

Subsequently, we propose our symbolic approximations together with lower-bounding distance measures. In contrast to the SAX extensions mentioned above, they provide a higher representation accuracy while having the same representation size as SAX.

6.2 SEASON- AND TREND-AWARE SYMBOLIC APPROXIMATION

As we observed in Chapter 4, time series from many domains such as meteorology, energy, and economy exhibit deterministic behavior. The wind speed has a yearly season, while the solar irradiation has a strong daily season. Consequently, this cyclically repeated behavior has an effect on the amount of energy production from renewables. In

energy consumption, human behavior leads to the observation of weekly seasons. Economic time series may exhibit a trend due to, for example, an increase in sales of a product.

Researchers have always applied SAX to datasets with trends and seasons, whether it was a synthetic [LKLC03, LKWL07] or a real-world dataset [SK08]. However, they did not exploit this deterministic behavior, which is why it affected all symbols redundantly and distorted the symbolic distribution. As a consequence, we argue that taking this behavior into account is essential for an accurate representation.

In this section, we propose sSAX and tSAX that are aware of the time series' season and trend, respectively. Each technique is described along with its time series model, its representation, its distance measure, and its properties. Moreover, both techniques take into account heuristics for improving the symbolic distribution. These heuristics are efficiently calculated in the preprocessing step which is required by SAX to normalize the time series.

6.2.1 Season-aware Symbolic Approximation

The sSAX technique is aware of a time series' season, i.e., it explicitly assumes a seasonal component in the time series. The remaining part of the time series forms the residuals.

Time Series Model

The use of the sSAX technique is restricted to a season-aware time series model:

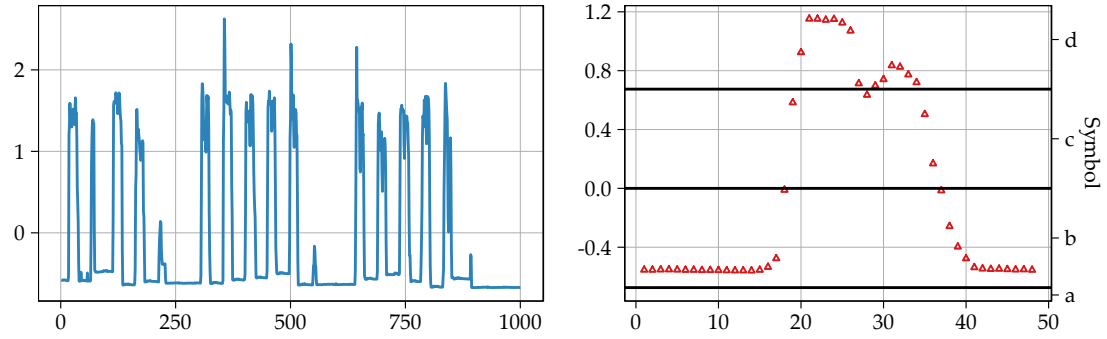
$$\underline{y} = \underline{seas} + \underline{res} \tag{6.9}$$

where $\underline{seas}$ is the seasonal and $\underline{res}$ is the residual component of the time series $\underline{y}$. It deviates from the more general form (Section 4.2, page 43) in two assumptions. First, it only assumes one seasonal component. Second, the season is extracted by averaging all values at the same seasonal position l [KS83] and not by multi-seasonal decomposition (Section 4.3, page 43). Through these assumptions, we are able to provide a lower-bounding distance measure. A seasonal feature σ_l ($1 \leq l \leq L$) is calculated as follows:

$$\sigma_l = \frac{L}{T} \sum_{k=1}^{T/L} y_{(k-1)\cdot L+l} \tag{6.10}$$

where T/L is the number of seasons in the time series that are iterated by k. The resulting features $\underline{\sigma}$ form the season mask.

Figure 6.2 illustrates a time series with a season that repeats after 48 values (Figure 6.2a). Averaging the 1st, 49th, ..., and (T-L+1)th value yields the seasonal feature of the 1st position. Averaging all positions results in the season mask (Figure 6.2b). Every seasonal feature of this mask is further discretized into an alphabet of size A, here $A = 4$.



(a) Time Series (b) Season Mask

Figure 6.2: Time Series with Season

Representation

Similar to SAX, the reduction of sSAX into the low-dimensional space is carried out in two steps: first, the season-aware PAA reduces it in the time domain, second, the season-aware SAX reduces it in the value domain.

The *season-aware PAA* (sPAA) combines the season mask and the PAA of the residuals in one representation. While PAA would ignore the season of the time series by taking the mean value of a segment, sPAA explicitly extracts this season beforehand. Formally, the sPAA representation is the vector:

$$\underline{\bar{y}}^{\intercal}_{sPAA} = (\sigma_1, \ldots, \sigma_l, \ldots, \sigma_L, \overline{res}_1, \ldots, \overline{res}_w, \ldots, \overline{res}_W) \tag{6.11}$$

where $W \cdot L$ divides T.

This representation is made of real values, which is why it is further reduced by the discretization of *season-aware SAX* (sSAX). Let $A_{seas}, A_{res} \in \mathbb{N}_{>0}$ be the sizes of two alphabets. Let $\underline{b}_{seas}$ and $\underline{b}_{res}$ be the respective vectors of breakpoints that split the real numbers into A_{seas} and A_{res} intervals. Then, the sSAX representation is the vector $\underline{\hat{y}}_{sSAX}$:

$$\underline{\hat{y}}^{\intercal}_{sSAX} = (\hat{\sigma}_1, \ldots, \hat{\sigma}_l, \ldots, \hat{\sigma}_L, \widehat{res}_1, \ldots, \widehat{res}_w, \ldots, \widehat{res}_W) \tag{6.12}$$

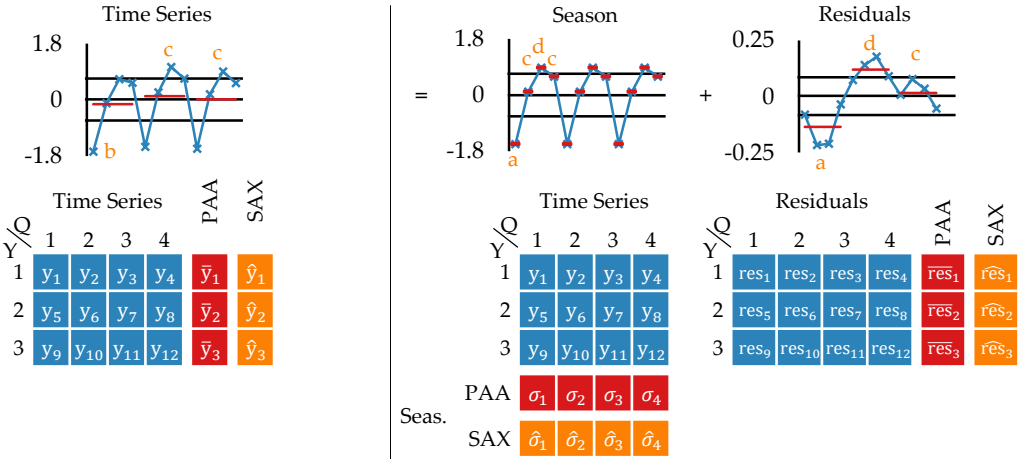
where $\hat{\sigma}_l$ is the symbol of σ_l discretized into the alphabet A_{seas} and $\widehat{res}_w$ is the symbol of $\overline{res}_w$ discretized into A_{res}.

Two heuristics retrieve the breakpoints. We quantify the influence of the season on the time series by the season strength R^2_{seas} (Equation 3.48, page 38). As mentioned earlier, this feature ranges between 0 and 1: $R^2_{seas} = 0$ means that the time series is determined by the residuals while $R^2_{seas} = 1$ shows a high influence of the season. Assuming that the residual and seasonal component are independent of each other, the following equations estimate the standard deviation of the season and the residuals:

$$sd(\underline{res}) = \sqrt{1 - R^2_{seas}} \tag{6.13}$$

$$sd(\underline{seas}) = \sqrt{1 - sd(\underline{res})^2} \tag{6.14}$$

where R^2_{seas} is the mean season strength of the dataset. Consequently, we set the breakpoints $\underline{b}_{seas}$ such that the area under normal distribution $\mathcal{N}(0, sd(\underline{seas}))$ is split into equiprobable regions $1/A_{seas}$.



(a) Original SAX (b) Season-aware SAX

Figure 6.3: Advantages of Season-aware SAX over SAX

Regarding the residuals, we also assume normally distributed mean values. After season extraction, the residual component has less influence and its variance does not reach one as assumed from Lin et al. Therefore, we set the breakpoints $\underline{b}_{res}$ such that the area under normal distribution $\mathcal{N}(0, sd(\underline{res}))$ is split into equiprobable regions $1/A_{res}$.

Figure 6.3 illustrates the advantages of sSAX over SAX using an economic time series, measured at a quarterly granularity (Q) during three years (Y). It displays the time series as lineplot and below, it arranges its values in an array. PAA and SAX (red and yellow column) only focus on a row-wise reduction of values, which cannot capture the cyclically repeated characteristics (Figure 6.3a). In contrast, sPAA and sSAX capture these characteristics first by a column-wise reduction, leading to the season mask (Figure 6.3b). Moreover, the residuals are reduced row-wise.

Distance

The techniques sPAA and sSAX provide distance measures $d_{sPAA}(\bar{\underline{y}}_{sPAA}, \bar{\underline{y}}'_{sPAA})$ and $d_{sSAX}(\hat{\underline{y}}_{sSAX}, \hat{\underline{y}}'_{sSAX})$:

$$d_{sPAA}(\bar{\underline{y}}_{sPAA}, \bar{\underline{y}}'_{sPAA}) = \sqrt{\frac{T}{W \cdot L}} \sqrt{\sum_{l=1}^{L} \sum_{w=1}^{W} (\sigma_l - \sigma'_l + \overline{res}_w - \overline{res}'_w)^2} \tag{6.15}$$

$$d_{sSAX}(\hat{\underline{y}}_{sSAX}, \hat{\underline{y}}'_{sSAX}) = \sqrt{\frac{T}{W \cdot L}} \sqrt{\sum_{l=1}^{L} \sum_{w=1}^{W} cell(\hat{\sigma}_l, \hat{\sigma}'_l, \widehat{res}_w, \widehat{res}'_w)^2} \tag{6.16}$$

The distance measure of sSAX relies on a lookup table that returns the precalculated distance of the season and residual symbols, using $\underline{b}_{seas}$ and $\underline{b}_{res}$ as breakpoints, respectively. However, this lookup table for four symbols may get huge, which is why we propose an equivalent formulation for two smaller lookup tables. Let c_s be a lookup table defined as follows:

$$c_s(a, a') = b_a - b_{a'+1} \tag{6.17}$$

where $\underline{b}_a$ are the breakpoints of the given feature. Then $cell(\hat{\sigma}, \hat{\sigma}', \widehat{res}, \widehat{res}')$ can be calculated by:

$$cell(\hat{\sigma}, \hat{\sigma}', \widehat{res}, \widehat{res}') = \begin{cases} c_s(\hat{\sigma}, \hat{\sigma}') + c_s(\widehat{res}, \widehat{res}') & c_s(\hat{\sigma}, \hat{\sigma}') \geq -c_s(\widehat{res}, \widehat{res}') \\ c_s(\hat{\sigma}', \hat{\sigma}) + c_s(\widehat{res}', \widehat{res}) & c_s(\hat{\sigma}', \hat{\sigma}) \geq -c_s(\widehat{res}', \widehat{res}) \\ 0 & \text{otherwise} \end{cases} \quad (6.18)$$

6.2.2 Trend-aware Symbolic Approximation

The tSAX technique is aware of the time series' trend and captures this behavior in a trend component.

Time Series Model

The use of the tSAX technique is restricted to a trend-aware time series model:

$$\underline{y} = \underline{tr} + \underline{res} \quad (6.19)$$

where $\underline{tr}$ and $\underline{res}$ are the trend and residual component, respectively. It deviates from the more general form (Section 4.2, page 43) in that it supports only a linear trend component. Trend changes and seasonal components are not included in order to provide the lower-bounding property.

Linear regression extracts the components from the time series. This technique is fast and allows for proving the lower-bounding property of subsequent distance measures. It estimates two features θ_1 and θ_2 which describe the base value and the slope of the time series, respectively. Consequently, the trend-aware time series model is equal to:

$$\underline{y} = \theta_1 + \theta_2 \cdot (\underline{t} - 1) + \underline{res} \quad (6.20)$$

where $\underline{t}^\intercal = (1, ..., t, ..., T)$ is the vector of time instances of the time series. Linear regression selects these features if they minimize the sum of squared residuals $\sum_{t=1}^{T} res_t^2$. Moreover, it yields two important properties. First, the sum of the residuals is always zero:

$$\sum_{t=1}^{T} res_t = 0 \quad (6.21)$$

and second, the deterministic components and the residuals are uncorrelated:

$$\sum_{t=1}^{T} (tr_t \cdot res_t) = 0 \quad (6.22)$$

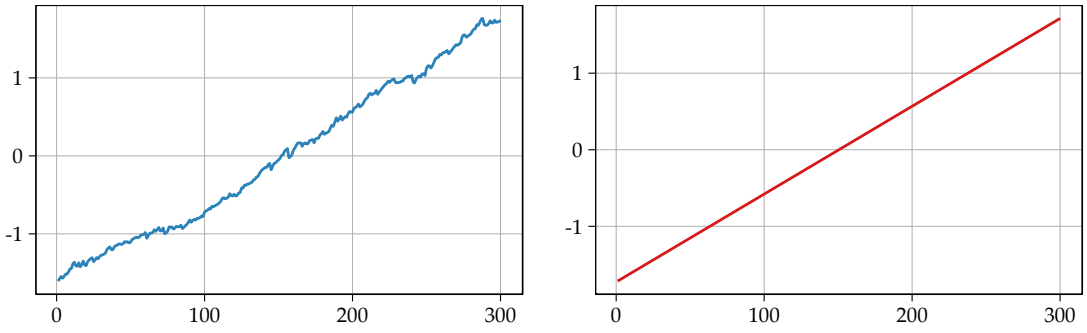
Figure 6.4 represents a time series with a strong trend component (Figure 6.4a). Its trend is extracted by linear regression (Figure 6.4b) and characterized by $\theta_1 = -1.72$ and $\theta_2 = 0.01$.

The features θ_1 and θ_2 are interdependent because the time series is normalized. Therefore, the following equation holds:

$$\theta_2 = -\frac{2}{T-1} \cdot \theta_1 \quad (6.23)$$

The interested reader finds the proof in the Appendix (Section A.3). Using this equation, θ_1 and θ_2 are combined to one *trend feature* ϕ that represents the angle between the x-axis and the trend component:

$$\phi = \arctan(\theta_2) \quad (6.24)$$



(a) Time Series (b) Season Mask

Figure 6.4: Time Series with Trend

Representation

Similar to SAX, the trend-aware symbolic approximation transforms a time series in two steps: first, the trend-aware PAA reduces the time series in the time domain; second, the trend-aware SAX reduces it in the value domain.

The *trend-aware PAA* (tPAA) representation is the vector of the trend feature and the mean values of the residual component:

$$\underline{\overline{y}}_{tPAA}^{\intercal} = (\phi, \overline{res}_1, \ldots, \overline{res}_w, \ldots, \overline{res}_W) \tag{6.25}$$

Let $A_{tr}, A_{res} \in \mathbb{N}_{>0}$ be the sizes of two alphabets. Let $\underline{b}_{tr}$ and $\underline{b}_{res}$ be the respective vectors of breakpoints that split the real numbers into A_{tr} and A_{res} intervals. The *trend-aware SAX* (tSAX) representation is the vector $\underline{\hat{y}}_{tSAX}$:

$$\underline{\hat{y}}_{tSAX}^{\intercal} = (\hat{\phi}, \widehat{res}_1, \ldots, \widehat{res}_w, \ldots, \widehat{res}_W) \tag{6.26}$$

where $\hat{\phi}$ is the symbol of ϕ discretized into the alphabet A_{tr} and $\widehat{res}_w$ is the symbol of $\overline{res}_w$ discretized into A_{res}.

Two heuristics retrieve the breakpoints. As a result of normalization, there is a minimum and a maximum feature ϕ that is reached if the time series is a perfect trend with zero residuals. Thus, ϕ is bounded by ϕ_{max}:

$$|\phi| \leq \phi_{max} \text{ where } \phi_{max} = \tan^{-1}\sqrt{1/var(\underline{t})} \tag{6.27}$$

Using this observation and the assumption that each trend is equiprobable, we set the breakpoints $\underline{b}_{tr}$ such that the area under uniform distribution between $[-\phi_{max}, \phi_{max}]$ is split into regions of probability $1/A_{tr}$.

Regarding the residuals, we adopt normally distributed mean values similar to SAX. After extracting the trend, the residual component has less influence. We quantify the influence of the trend on the time series by the trend strength R_{tr}^2. As mentioned earlier (Equation 3.47, page 3.47), this feature ranges between 0 and 1: $R_{tr}^2 = 0$ means that the time series is determined by the residuals while $R_{tr}^2 = 1$ shows a high influence of the trend. Assuming that the trend strength of the dataset is known and the time series are normalized, the standard deviation of the residuals is estimated by:

$$sd(\underline{res}) = \sqrt{1 - R_{tr}^2} \tag{6.28}$$

where R_{tr}^2 is the mean trend strength of the dataset. Thus, we set $\underline{b}_{res}$ such that the area under normal distribution $\mathcal{N}(0, sd(\underline{res}))$ is split into equiprobable regions $1/A_{res}$.

Distance

The techniques tPAA and sPAA provide the distance measures $d_{tPAA}(\overline{\underline{y}}_{tPAA}, \overline{\underline{y}}'_{tPAA})$ and $d_{tSAX}(\hat{\underline{y}}_{tSAX}, \hat{\underline{y}}'_{tSAX})$ that are given as follows:

$$d_{tPAA}(\overline{\underline{y}}_{tPAA}, \overline{\underline{y}}'_{tPAA}) = \sqrt{\sum_{t=1}^{T} (\Delta\theta_1 + \Delta\theta_2 \cdot (t-1) + \Delta\overline{res}_{\lfloor (t-1)/(T/W) \rfloor + 1})^2} \tag{6.29}$$

$$d_{tSAX}(\hat{\underline{y}}_{tSAX}, \hat{\underline{y}}'_{tSAX}) = \sqrt{c_t(\hat{\phi}, \hat{\phi}')^2 + \frac{T}{W} \sum_{w=1}^{W} cell(\widehat{res}_w, \widehat{res}'_w)^2} \tag{6.30}$$

where $\Delta f = f - f'$ is a shorthand symbol for the difference between a feature or a segment value of time series y and y'. The tSAX distance measure relies on a lookup table c_t for the trend feature using $\underline{b}_{tr}$ as breakpoints. It expresses the minimum distance of two trend components represented by $\hat{\phi}$ and $\hat{\phi}'$. For the residuals, tSAX relies on the lookup table $cell$ from SAX using $\underline{b}_{res}$ as breakpoints (Equation 6.8).

6.2.3 Properties of Engineering Techniques

We review sSAX and tSAX regarding their properties from Table 6.1.

Representation Size The alphabets A_{seas}, A_{tr}, and A_{res} are chosen such that the representation size of sSAX and tSAX equals the representation size of SAX. If they are not a power of 2, we allow for interleaving as in [MGQT13].

Representation Time For sSAX, the representation needs one pass over the time series because the season mask and the residuals can be calculated simultaneously. The tSAX representation needs an additional pass for the linear regression.

Distance Storage Both representations need two lookup tables of size A_{res}^2 and either A_{seas}^2 or A_{tr}^2. Depending on the alphabet sizes, this leads to a storage for the distance calculation that is smaller or larger compared to SAX.

Distance Time The sSAX distance measure needs at most $4 \cdot W \cdot L$ lookups instead of W lookups due to the combinations of season and residual symbols. Although sSAX may use fewer segments for the residuals than SAX does for the time series, it leads to more lookups. However, if the calculation of the Euclidean distance is much slower since it incurs disk I/O, this limitation can be accepted. The tSAX distance measure needs only 1 lookup for the trend and W lookups for the residuals.

Lower-bounding Distance The most remarkable finding is that all presented distance measures, d_{sPAA}, d_{tPAA}, d_{sSAX}, and d_{tSAX} lower-bound the Euclidean distance measure. The interested reader finds the proofs in the Appendix (Sections A.1, A.2, A.4, and A.5).

6.3 EXPERIMENTAL EVALUATION

We compare our techniques sSAX and tSAX to the competitors SAX and 1d-SAX in order to examine the following hypotheses: first, they improve the symbolic distribution of the residuals, second, they provide a higher representation accuracy, and third, they allow for a more accurate and efficient time series matching. We begin by giving our experimental setting. Subsequently, we present and discuss the results.

6.3.1 Experimental Setting

This section details the experimental setting, i.e., the matching methods that are applied, the time series datasets that are selected, the output variables that are measured, the configurations chosen for the engineering techniques, as well as the software and hardware environment.

Matching Methods

Besides the evaluation of the representation accuracy, we assess the time series representations used in two matching methods: the exact and approximate time series matching.

Exact matching returns the observation from a time series dataset that has the minimum Euclidean distance to the query time series. It conducts a *linear search*: First, the representation distance of the query to each observation in the dataset is calculated. These distances are sorted increasingly. Second, the Euclidean distance from the observations is calculated in the order of their representation distance, keeping track of the "best-so-far" observation and its Euclidean distance. If this "best-so-far" (Euclidean) distance is less than the representation distance of the next observation, the linear search terminates and returns the "best-so-far" observation as an exact match. The early termination is possible because of the lower-bounding property: subsequent observations never have a Euclidean distance that is smaller than the "best-so-far" distance.

Approximate matching returns the observation that is almost as good as the exact match. It conducts a linear search on the representation, too. However, it returns the observation with the minimum representation distance as an approximate match. If there is more than one observation with the minimum representation distance, it returns the observation with the minimum Euclidean distance within this set.

Datasets

To study the behavior of the engineering techniques, they are evaluated on synthetic time series datasets with specific characteristics and on two real-world time series datasets. Table 6.2 recaps the dataset dimensions.

Season A Season dataset contains 1,000 random walk time series, each of which is overlaid with a season mask of length 10. In compliance with [SK08], the time series length varies between 480 and 1,920. All time series of a dataset have the same season strength which is fixed to a value between 1 and 99%, where a tolerance of 0.5 percentage points (pp) in both directions is accepted.

Trend A Trend dataset contains 1,000 random walk time series, each of which is overlaid with a trend. Similarly to the Season datasets, the time series length varies between 480 and 1,920. All time series of a dataset have the same trend strength between 1 and 99% with a tolerance of 0.5 pp.

Metering (1.5 years) As described in Section 4.1 (page 42), the Metering dataset is a real-world dataset with seasonal components. The season-aware SAX is evaluated with respect to the daily season, which has, on average, a season strength of 18.3%. In contrast to the presentation in Chapters 4 and 5, we do not focus on the yearly season; thus, we focus on the original 1.5 years.

Table 6.2: Dataset Dimensions

Dataset	Dataset Size	Time Series Length
Season	1,000	[480; 960; 1,440; 1,920]
Trend	1,000	[480; 960; 1,440; 1,920]
Metering	5,958	21,840
Economy (M4)	6,400	300
Season (Large)	[6,510,417; 13,020,833]	960

Economy (M4) The Economy (M4) dataset contains about 100,000 time series from different domains (finance, industry, demography, macro-economy, other). It is the result from the M4-Competition to systematically evaluate the accuracy of forecast techniques on a defined dataset [MSA18]. The values of each time series have a specified interval (year, quarter, month, other) and exhibit a trend component. In compliance with our dataset definition, only time series with the same length are selected, i.e., 6,400 time series measured for 25 years with monthly granularity. In the scope of this chapter, we refer to this dataset as Economy dataset, and omit the name of the competition.

Season (Large) For the efficiency evaluation, two Season datasets with an overall size of 50 and 100 Gb are included. The time series length is fixed to 960 values. In contrast to Season, the season strength of a time series may vary. We select datasets such that their season strength is on average 10.0% (weak), 50.0% (medium), and 90.0% (strong).

For the accuracy evaluation, each time series from a dataset acts as query and is matched against the dataset of the remaining time series. Thus, there are as many queries as there are time series in the dataset. For the efficiency evaluation on Season (Large), we randomly select up to 50 query time series for each dataset. We limit an experiment to four hours. Since the runtime differs for each query, each technique is evaluated with the same set of queries.

Output Variables

Five output variables assess the accuracy and efficiency of symbolic approximations and enable us to evaluate our hypotheses: the *entropy*, the *tightness of lower bound*, the *pruning power*, the *approximate accuracy*, and the *runtime* which are defined subsequently.

Entropy We hypothesize that sSAX and tSAX improve the distribution of the residual symbols compared to the SAX symbols. A uniform distribution is a desirable property as Butler and Kazakov point out [BK15]. This property is quantified by the entropy as follows:

$$H(A) = -\sum_{1 \leq a \leq A} freq_a \cdot \mathrm{ld}(freq_a) \tag{6.31}$$

where $freq_a$ is the relative frequency of a residual symbol in a time series dataset. An equal frequency of all symbols leads to the maximum entropy. If some symbols are more frequent than others, the entropy decreases. Only the entropy of two alphabets with equal size are compared since alphabets with different sizes have a different maximum entropy.

Tightness of Lower Bound Our second hypothesis is that sSAX and tSAX provide a higher representation accuracy than the competitors. This is evaluated with the tightness of lower bound (TLB) in accordance with [LKLC03]. The TLB expresses the ratio between the representation distance and the Euclidean distance as follows:

$$TLB(\underline{y}, \underline{y}') = \frac{d_{*SAX}(\hat{\underline{y}}, \hat{\underline{y}}')}{d_{ED}(\underline{y}, \underline{y}')} \tag{6.32}$$

where d_{*SAX} is either d_{SAX}, d_{1d-SAX}, d_{sSAX}, or d_{tSAX}. To evaluate the TLB of a time series dataset, the mean TLB of all time series combinations is calculated.

Accurate and Efficient Time Series Matching Our third hypothesis is that time series matching with sSAX and tSAX is more accurate and efficient than the competitors. Accuracy it is evaluated with the pruning power or the approximate accuracy, depending on the matching method.

Exact matching is improved if more observations can be pruned and the linear search terminates earlier. The pruning power (PP) expresses the fraction of observations that can be pruned and whose Euclidean distance is not evaluated [CCL+07]. A pruning power of 0 means that no observations are pruned, a pruning power close to 1 means that the linear search terminates after the first observation.

Approximate matching is improved if the approximate match is closer to the Euclidean distance of the exact match. We introduce the output variable approximate accuracy (AA) which is the quotient of the Euclidean distance between the query and the exact match and the Euclidean distance between the query and the approximate match. An approximate accuracy of 0 means that the approximate match is very inaccurate, an approximate accuracy of 1 means that the approximate match is as accurate as the exact match.

The efficiency of time series matching is evaluated with the runtime. We measure the wall-clock time in seconds for the calculation of the representation distances and the Euclidean distances.

Configurations

Table 6.3 summarizes all possible configurations for the number of segments W and the alphabet size A for each dataset and each engineering technique. The representation size is fixed, which is why the alphabet A_{res} of sSAX and of tSAX is set in accordance to A_{seas} and A_{tr}, respectively. 1d-SAX uses the alphabet A_a for the base value of a segment and the alphabet A_s for the slope [MGQT13]. Alphabet sizes less or equal than 4 are ignored since they evidently cause a high accuracy loss. We limit the size of a lookup table to $4\ Mb$, which corresponds to an alphabet of size $1{,}024$. The standard deviation of sSAX and tSAX to discretize the residuals is derived from the component strength (Equations 6.13 and 6.28).

Software and Hardware Environment

All engineering techniques are implemented as an R package and published together with the scripts and datasets of this evaluation [Keg19a, R C18]. For the runtime evaluation, matching methods are implemented in C and compiled with GCC 6.3.0 with level 3 optimization under Windows 10. Experiments run on a machine with Intel(R) i7 Processor 6660U@2.60GHz and 20 Gb of RAM and compare two disks, one 1 Tb HDD and one 500 Gb SSD. Each time series is stored as a binary file on disk. Time series representations and lookup tables are kept in-memory, while time series are read from disk without system cache buffering.

Table 6.3: Configurations of Engineering Techniques

Synthetic	W	A or A_{res}	A_{seas} or A_{tr}	Size (bit)
SAX	[32; 40; 48; 96]	[1,024; 256; 101; 10]	–	320
sSAX	[24; 48; 48]	[1,024; 32; 64]	[256; 256; 9]	320
tSAX	[32; 40; 48; 96]	$\lfloor 2\hat{}((320 - \text{ld}(A_{tr}))/W) \rfloor$	[32; 128; 1,024]	320
Metering	**W**	**A or A_{res}**	**A_{seas} or A_{tr}**	**Size (bit)**
SAX	[455; 520; 728; 910]	[256; 128; 32; 16]	–	3,640
sSAX	455	[191; 165; 142; 123]	[16; 64; 256; 1024]	3,640
Economy	**W**	**A, A_a, or A_{res}**	**A_{tr} or A_s**	**Size (bit)**
SAX	[10; 12; 15; 20; 30]	[256; 101; 40; 16; 6]	–	80
1d-SAX	[10; 12; 15; 20]	$\lfloor 2\hat{}((80 - \text{ld}(A_s) \cdot W)/W) \rfloor$	[8; 16; 32]	80
tSAX	[10; 12; 15; 20; 30]	$\lfloor 2\hat{}((80 - \text{ld}(A_{tr}))/W) \rfloor$	[16; 64; 256; 1,024]	80

6.3.2 Results and Discussion

The results of our evaluation are presented and discussed in the order of our hypotheses. The quality of the symbolic distribution is assessed, followed by the evaluation of the representation accuracy. Finally, the accuracy and efficiency of matching methods are evaluated.

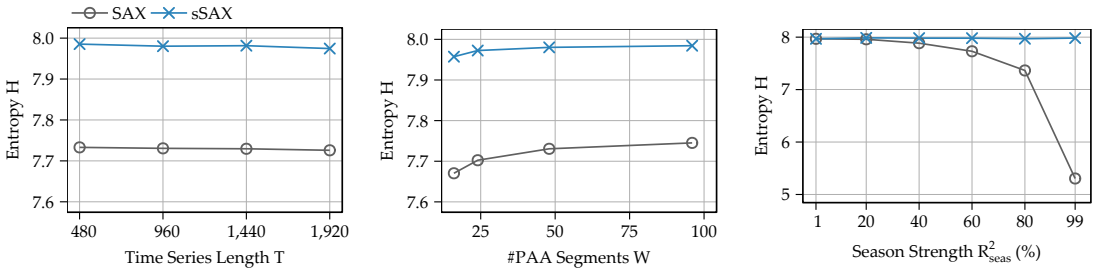
Symbolic Distribution

We compare the symbolic distribution of SAX, sSAX, and tSAX utilizing the entropy. For this evaluation, the alphabet size is the same for all configurations and is fixed to $A = A_{res} = 256$ which results in a maximum entropy of $H_{max}(A) = 8$.

Figure 6.5 visualizes the entropy of SAX and sSAX on the Season datasets. The sSAX technique systematically provides a higher entropy, which results in more equally distributed symbols. The entropy decreases if time series get longer (Figure 6.5a) or if there are fewer segments, i.e., longer segments (Figure 6.5b). This observation confirms the observation of Butler and Kazakov that the mean value distorts the symbolic distribution [BK15]. However, this effect is less strong for sSAX. Interestingly, the entropy of SAX significantly decreases for datasets with a strong season, but it is not the case for sSAX (Figure 6.5c). This finding confirms that the seasonal component should be treated separately.

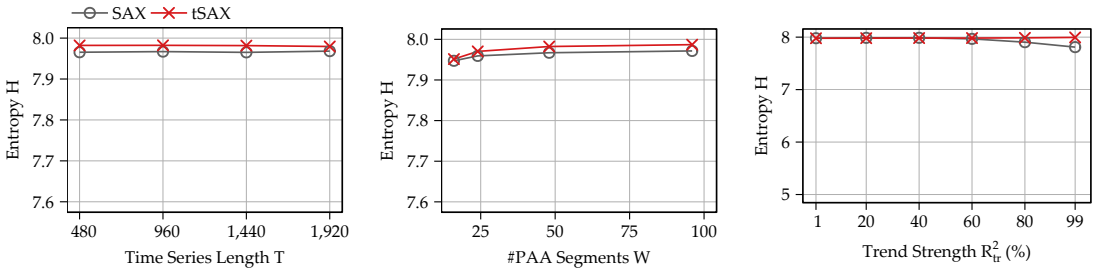
Figure 6.6 visualizes the entropy of SAX and tSAX on the Trend datasets. Figures 6.6a and 6.6b reconfirm the finding that the mean value distorts the symbolic distribution and that tSAX remediates this effect. Moreover, Figure 6.6c shows that the entropy decreases if SAX is applied to a time series with strong trends and that this entropy is more stable for tSAX. Overall, both techniques exhibit a high entropy close to the maximum. Therefore, the gain of tSAX over SAX is less strong than the gain of sSAX over SAX.

On the real-world dataset, we compare the techniques for the same number of segments W. On the Metering dataset, the entropy increases from 6.96 for SAX to 7.09 for sSAX. On the Economy dataset, the entropy increases from 7.92 for SAX to 7.95 for tSAX. The overall increase of entropy is less strong compared to the homogeneous synthetic datasets. This result is due to the heterogeneous component strengths in both datasets, and due to the heuristics which assume the mean component strength.



(a) By Series Length (b) By Number of Segments (c) By Season Strength

Figure 6.5: Entropy on Season



(a) By Series Length (b) By Number of Segments (c) By Trend Strength

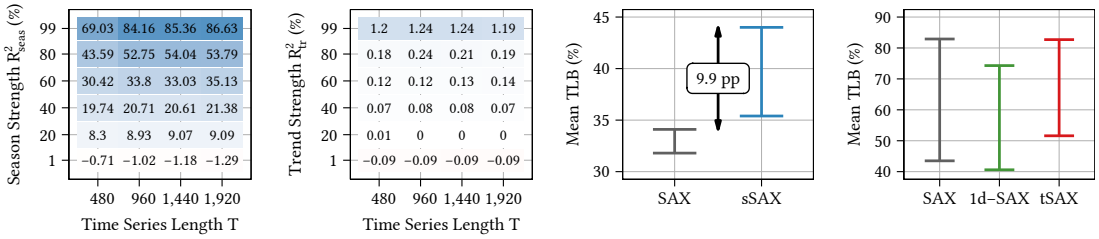
Figure 6.6: Entropy on Trend

Representation Accuracy

We evaluate the representation accuracy utilizing the TLB (Figure 6.7). For this evaluation, the representation size is constant for each dataset, and the possible configurations are given in Table 6.3.

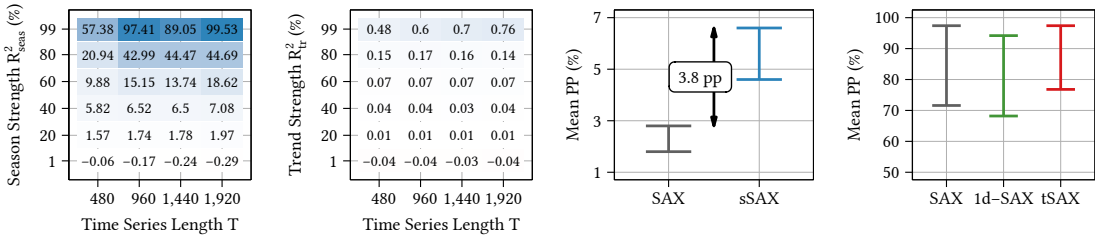
On the synthetic datasets, the TLB of SAX is compared to sSAX and tSAX (Figures 6.7a and 6.7b). Results are grouped by time series length and component strength. Each cell presents the difference in percentage points between the mean TLB of the most accurate sSAX/tSAX configuration and the mean TLB of the most accurate SAX configuration. Figure 6.7a shows that sSAX gains accuracy compared to SAX with up to 86 pp. The longer the time series and the stronger the season, the higher is the accuracy gain. If there is no season, sSAX is only slightly less accurate. The tSAX technique gains accuracy by only 1.2 pp and has slight losses in the absence of a trend (Figure 6.7b). This gain is lower than expected. However, there are two possible reasons for this. First, SAX satisfactorily captures the global trend of a time series. Second, the normalization transforms the time series such that the trend has less influence. Therefore, tSAX has not much room for improvement.

Figures 6.7c and 6.7d display the results for the real-world datasets. It shows the minimum and maximum mean TLB which are reached with the chosen configurations. On the Metering dataset, the best sSAX configuration gains up to 9.9 pp compared to the best SAX configuration (Figure 6.7c). Thus, if the season is taken into account, it leads to a much higher representation accuracy. On the Economy dataset, we include 1d-SAX in our comparison, which is the only trend-aware SAX extension that has the same representation size as SAX. Overall, tSAX has a better representation accuracy than 1d-SAX. Thus, it better takes advantage of the available representation size. However, it does not gain compared to the best SAX configurations and reaches at best 82.7% while SAX reaches 82.9% (Figure 6.7d).



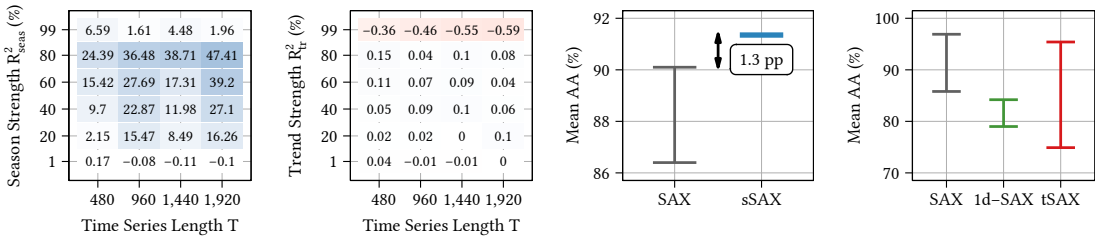
(a) Season (b) Trend (c) Metering (d) Economy

Figure 6.7: Increase of TLB compared to SAX



(a) Season (b) Trend (c) Metering (d) Economy

Figure 6.8: Increase of Pruning Power Compared to SAX



(a) Season (b) Trend (c) Metering (d) Economy

Figure 6.9: Increase of Approximate Accuracy Compared to SAX

Exact Matching

For exact matching, we first evaluate the pruning power that results from the representation accuracy (Figure 6.8).

On the synthetic datasets, sSAX and tSAX exhibit a gain in pruning power compared to SAX. Remarkably, sSAX improves the pruning power up to 99 pp in the presence of a strong season (Figure 6.8a). However, if no season is present, sSAX has a worse pruning power by at most 0.29 pp which is negligible regarding the overall gain of sSAX. The tSAX technique improves the pruning power even for weak trends (Figure 6.8b). But the gain is limited with at most 0.76 pp.

This behavior is also confirmed on the real-world datasets. On Metering, sSAX gains 3.8 pp in pruning power and reaches 6.6% (Figure 6.8c). While SAX can prune 274 from 5,958 time series on average, sSAX efficiently prunes 393 time series. On Economy, the best SAX configuration already has a very high pruning power (Figure 6.8d) with 97.4%. The tSAX technique outperforms 1d-SAX but reaches at best the same pruning power as SAX.

Table 6.4: Matching Efficiency on Season (Large)

HDD Size	Technique	Repr.	$R^2_{seas} = 10.0\%$ Raw	$R^2_{seas} = 10.0\%$ Sum	$R^2_{seas} = 50.0\%$ Raw	$R^2_{seas} = 50.0\%$ Sum	$R^2_{seas} = 90.0\%$ Raw	$R^2_{seas} = 90.0\%$ Sum
50 Gb	SAX	1.80	135.82	137.61	1,801.12	1,802.92	6046.75	6048.55
	sSAX	8.67	41.76	**50.43**	0.54	**9.21**	0.08	**8.75**
100 Gb	SAX	3.69	73.61	77.30	4181.02	4184.72	13,423.47	13,427.16
	sSAX	16.86	4.77	**21.63**	1.09	**17.95**	0.11	**16.97**
SSD Size	**Technique**	**Repr.**	$R^2_{seas} = 10.0\%$ **Raw**	$R^2_{seas} = 10.0\%$ **Sum**	$R^2_{seas} = 50.0\%$ **Raw**	$R^2_{seas} = 50.0\%$ **Sum**	$R^2_{seas} = 90.0\%$ **Raw**	$R^2_{seas} = 90.0\%$ **Sum**
50 Gb	SAX	1.84	4.05	**5.89**	101.61	103.45	850.81	852.65
	sSAX	9.12	0.71	9.83	0.04	**9.16**	0.02	**9.14**
100 Gb	SAX	3.80	8.29	**12.09**	115.14	118.95	1,088.80	1,092.60
	sSAX	17.99	1.05	19.04	0.07	**18.06**	0.02	**18.02**

Let us now look at the efficiency evaluation. Table 6.4 details the runtimes for both disks (HDD, SSD) on the Season (large) datasets with 50Gb and 100Gb. The runtime is broken down into one part for calculating the representation distances including result ordering (Repr.), and in another part for accessing the time series and calculating the true distances (Raw). Although the representation times are not affected by the underlying disk, the experiments provide slightly different runtimes due to measuring inaccuracy. For each season strength, the sum of both parts indicates the mean runtime per query. Calculating the representation distances is displayed once for all season strengths since it does not depend on this heuristic.

The table reveal that (1) sSAX is faster for all datasets from HDD even when there is only a weak season strength, (2) sSAX is faster for all datasets from SSD for a significant season strength. The most striking result to emerge from the data is that sSAX is up to three orders of magnitude faster for time series with a strong season. On HDD, sSAX requires approximately 17 seconds for querying the 100 Gb dataset, while SAX requires approximately 3.7 hours. SAX has a decreased pruning power and thus, needs more disk access. The pruning power of sSAX, however, increases and provides exact matches even faster. A naive matching of the time series as is (raw-data-based) using the Euclidean distance would require 6,137 seconds, i.e., approximately 53 minutes (50 Gb on SSD), and 13,866 seconds, i.e., 231 minutes (100 Gb on SSD); thus it is much slower than sSAX. We did not evaluated naive matching on HDD due to time restrictions.

Approximate Matching

In approximate matching, we evaluate the accuracy of an approximate match compared to the exact match utilizing the approximate accuracy (Figure 6.9).

Figures 6.9a and 6.9b show the increase of approximate accuracy on the synthetic datasets by time series length and component strength. The longer the time series and the stronger the deterministic component, the higher is the gain of sSAX and tSAX over SAX. The sSAX reaches up to 47 pp improvement on Season, i.e., the approximate match is, on average, 50% more accurate (Figure 6.9a). Due to the aforementioned reasons, tSAX only reaches minor improvements on Trend with up to 0.14 pp (Figure 6.9b). On datasets with a strong component strength, both representations reach an approximate accuracy of approximately 99%. The sSAX and tSAX representations reach this accuracy thanks to the accurate representation. SAX reaches this accuracy because most of the observations

have the same representation which is why SAX re-evaluates their Euclidean distance to the query in order to retrieve the most accurate approximate observation. Thus, it reaches a slightly higher approximate accuracy, which is mainly due to the evaluation of the Euclidean distance.

On the real-world datasets, sSAX and tSAX show a similar behavior. All sSAX configurations outperform all SAX configurations on Metering, and the best sSAX configuration is up to 1.3 pp more accurate than the best SAX configuration (Figure 6.9c). The exact search shows that the time series are all very close to each other regarding the Euclidean distance. The pruning power of 6.6% shows that for many observations, it has to be checked whether they are the exact match (Figure 6.8c). However, the approximate match already reaches approximately 91.5% of the accuracy of the exact match. Although tSAX provides a higher approximate accuracy compared to 1d-SAX (95.4% vs. 84.2%), it cannot reach the best approximate accuracy of SAX (96.9%) (Figure 6.9d).

Let us now look at the efficiency evaluation. For all chosen queries, there are not two matches with the same minimum representation distance (which would require the calculation of the Euclidean distance). Therefore, we can focus on the representation distance calculation. Table 6.4 reveals that approximate matching with sSAX is slower compared to SAX due to an increased number of lookups. However, these approximate matches are much more accurate, as Figure 6.9 suggests.

6.4 SUMMARY

Based on our feature-based engineering technique, we have devised two novel symbolic approximations, sSAX and tSAX. They extend the shape-based technique SAX by the trend feature and the season mask, respectively, and thus, they take an important behavior of time series into account. Compared to other SAX extensions, they combine characteristics from segments with global characteristics that affect the full time series. Moreover, they do not increase the representation size and keep a comparable representation time and distance storage. Most importantly, they provide lower-bounding distance measures. Overall, they improve the symbolic distribution, the representation accuracy, and the time series matching compared to state-of-the-art techniques. Especially sSAX makes automation and data-mining tasks much more efficient in domains where deterministic behavior arises.

7

TIME SERIES CLASSIFICATION

TIME SERIES CLASSIFICATION differs greatly from the aforementioned data-mining tasks. It assumes that a dataset contains time series from several *classes* and that they are annotated by a *class label*. The task is to estimate a model, a so-called *classifier*, which maps time series from a training dataset to a class. Subsequently, this classifier is used to infer the class of time series from a test dataset.

Time series classification is desirable in many domains, often for supporting human classification decisions, or for fully automatizing processes. For example, image classification is a crucial task in computing, and authors suggest to extract image properties as time series. A classifier is estimated on this data helping users to classify the images [YK09]. In medicine, classifiers support physicians for their ECG analysis, as we discussed earlier (Section 2.1, page 15). In other circumstances, classifiers are fully automatized; human interaction is not required anymore, or even not possible. This arises, for example, from speech sentiment analysis in human-computer interaction [DPW96], from consumer profiling and target marketing [Agg02], and from anomaly or risk detection of monitored processes [NAM01, WGH12, FJ14, HWN15].

A multitude of engineering techniques has been proposed for time series classification, from raw-data-based [BLB+17], shape-based [DTS+08, YK09, BLB+17], to model- and feature-based techniques [XPK10]. They are evaluated on datasets from several domains [CKH+15, BLB+17]. While these datasets have a considerable size of up to 24,000 time series, the length of these time series is rather short, ranging from between 24 and 2,844. Thus, these works do not focus on the current challenges that we observe in many domains (Section 2.3): time series are captured at a fine granularity, which makes classification a challenging task, especially regarding efficiency.

Our objective is the classification of datasets with long time series, i.e., consisting tens of thousands of values. We assess *classification techniques* regarding their effectiveness, i.e., their ability to classify unlabeled time series accurately, but most importantly, we assess their efficiency, i.e., the runtime they need to represent a dataset, and to estimate and use a classifier. Various techniques have been proposed to increase the effectiveness of classification. Carried out on short time series, they can accept a quadratic time complexity or even worse [BLB+17]. The most accurate classification technique, *COTE*, has a worst-case complexity $\mathcal{O}(I^2T^4)$, where I is the size of the time series dataset and T is the length of the time series. However, this is clearly infeasible for big time series datasets as our potential analysis of COTE suggests (Table 7.1). A single-threaded classifier estimation of the Metering dataset on our hardware would require 16 days if the time series were short (Table 7.1a). If the time series were two years long, it would take thousands of years (Table 7.1b).

Table 7.1: Model Training Runtime of COTE

(a) Time Series Length $T = 60$

Dataset Size	Runtime
10	6.7 s
20	16.0 s
30	31.4 s
40	53.2 s
100	4.6 m
300	1.2 h
4,871	*16 d*

(b) Dataset Size $I = 50$

Time Series Length	Runtime
20	12.0 s
40	36.8 s
60	1.4 m
80	2.4 m
160	9.6 m
480	3.0 h
35,040	*11,772 y*

Clearly, we have to rely on more efficient techniques for classifying long time series. While one work is based on a shape-based representation [BJ14], most of the approaches focus on feature-based representations [Mör03, FJ14, CBNKL18] and a subsequent feature selection. However, these features are very general and not always efficiently calculated. Christ et al. provide a comprehensive feature vector with more than 700 features, providing potentially highly discriminative features. However, calculating all these features takes a considerable amount of runtime.

With this in mind, we propose a feature-based classification based on our representation from Chapter 4. Not only is this technique tailored to the seasonal and trend effects ocurring in a multitude of domains but the features are also efficiently calculated. We show experimentally on two real-world datasets that it competes and even outperforms most accurate classifiers, while keeping a reasonable overall runtime for representation and classification.

This chapter is structured as follows. We review existing classification techniques used for big time series datasets in Section 7.1. In Section 7.2, we give a system overview of our classification technique based on our feature-based representation, along with normalization techniques, and feature selection techniques. We evaluate our technique and its competitors in Section 7.3 regarding effectiveness and efficiency, and summarize our observations in Section 7.4.

7.1 STATE OF THE ART

There are *eager* and *lazy* classification techniques. An eager technique estimates a classifier based on the training dataset and captures its discriminative characteristics as a model. After estimation, this classifier is used on an unlabeled test dataset. In contrast, a lazy technique uses the training dataset for classification without an intermediate model. While it avoids the runtime for classifier estimation, it may require more time for inferring the class label than an eager technique.

Authors suggest reducing a time series to a low-dimensional representation and presenting it to standard classification techniques, rather than constructing complex classification technique for a high-dimensional space [BDHL12]. With this in mind, we review three engineering techniques that have been proposed to classify long time series, the run length distribution (Subsection 7.1.1), the discrete wavelet transform (Subsection 7.1.2), and a large feature vector (Subsection 7.1.3).

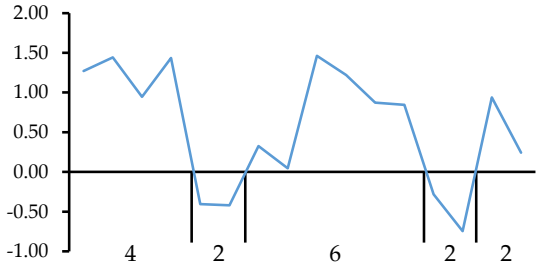


Figure 7.1: Run Length Distribution

Table 7.2: Haar Wavelet

Granularity	Mean Values	Wavelet Features
$k = 4$	$\underline{y}^\intercal = (8, 6, 2, 3, 4, 6, 6, 5)$	–
$k = 3$	$(7, 2.5, 5, 5.5)$	$(\mathbf{1}, \mathbf{\text{-}0.5}, \mathbf{\text{-}1}, \mathbf{0.5})$
$k = 2$	$(4.75, 5.25)$	$(\mathbf{2.25}, \mathbf{\text{-}0.25})$
$k = 1$	$(\mathbf{5})$	$(\mathbf{\text{-}0.25})$

7.1.1 Run Length Distribution

Bagnall and Janacek present the *run length distribution*, which is a shape-based engineering technique [BJ14]. A run length is the length of a time series subsequence between two zero crossings, i.e., a subsequence that is completely on one side of the x-axis. The run length distribution counts all run lengths that occur in a time series. Figure 7.1 represents a time series of length 16. The run length 2 occurs three times, and the run lengths 4 and 6 occur once, forming a distribution (0, 3, 0, 1, 0, 1). Since run lengths without occurrence are pruned it results in a run length distribution (3, 1, 1). Bagnall and Janacek perform a 1NN classification based on the distance between two run length distributions and show the superiority of dynamic time warping (Section 3.1, page 24) compared to other distance measures experimentally.

7.1.2 Discrete Wavelet Transform

The discrete wavelet transform is a feature-based technique from the frequency domain and is applied for classification in several works [Agg02, Mör03, MDC19]. We select this transform and not a Fourier transform because of its higher efficiency, as noted in Section 3.4 (page 36). It decomposes a time series into features at different levels of granularity, capturing global time series properties as well as local ones. Its simplest, yet effective configuration is the *Haar wavelet*, which is explained in Table 7.2, using the example from [Agg02]. For a given time series $\underline{y}$ of length 8, neighbored values are successively averaged, $(8 + 6) \ / \ 2 = 7$, leading to the mean values of the next granularity, $k = 3$. The distance of a mean value to its components, $8 - 7 = -(6 - 7) = 1$, is the wavelet feature. The time series can be fully reconstructed using the wavelet features and the time series mean value from the last granularity ($k = 1$). Thus, the representation is $(5, -0.25, 2.25, -0.25, 1, -0.5, -1, 0.5)$, which is printed in boldface. The resulting feature-based representation is still as long as the time series itself. Feature selection is carried out by the classifier [Agg02] or by a heuristic prior to classification [Mör03]. Finally, an eager classifier is estimated on the selected features [Mör03].

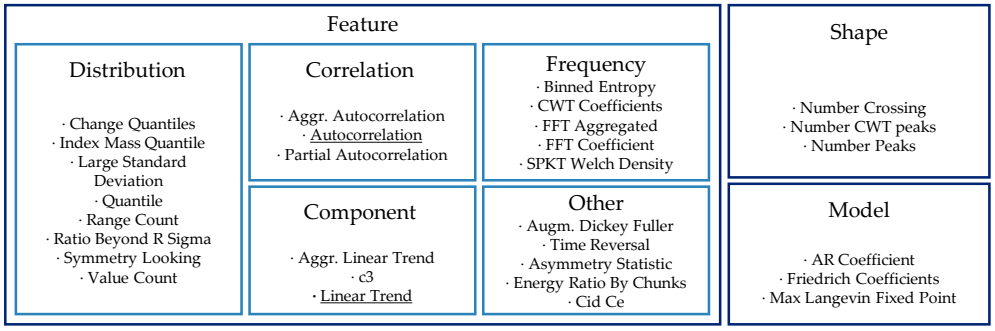


Figure 7.2: Features from tsfresh

7.1.3 Large Feature Vector

While the aforementioned techniques rely on specific shapes or features of a time series that are discriminative for long time series, Christ et al. and Fulcher et al. decide to take features from many categories into account, forming a large feature vector [CKLF16, FJ14]. Christ et al. names its feature vector *tsfresh*, consisting of more than 700 features. As we will explain in our experimental setting, we only include features that the authors define as comprehensive and efficient. These features are illustrated in Figure 7.2, further details can be found in [CKLF16]. Based on our five feature categories (Section 3.4, page 36), we can classify most of the features as distribution, correlation, component, and frequency features. Besides, the authors include other features as well as characteristics from shape- and model-based representation that are efficiently calculated. Among these features, only the autocorrelation and the linear trend corresponds to our feature vector; the season masks are not included. The moment features belong to the full feature vector of Christ et al., but not to the comprehensive and efficient subset. Based on this feature vector, discriminative features are selected, and an eager classifier is estimated. The authors propose their own feature selection technique, which is based on hypothesis tests [CKLF16].

The large feature vector of Fulcher et al. [FJ14] is highly similar to tsfresh. But it consists of thousands of features, which takes even more time than tsfresh to represent a time series. Consequently, we focus on tsfresh as competitor of this group.

7.2 SYSTEM OVERVIEW

In this section, we explain the techniques that we apply for our feature-based classification. To highlight the major steps, Figure 7.3 shows an overview. A *labeled dataset* contains time series with class labels. We shortly formalize this dataset in Subsection 7.2.1. Each time series is *decomposed* and reduced to a *feature-based representation*. In Subsection 7.2.2, we explain how we apply our feature-based engineering technique to this data-mining task. Subsequently, these features are *normalized* to the same range. We present normalization techniques in Subsection 7.2.3. Although our feature vector is already small, not all features are highly discriminative. Therefore, we present a *feature selection* in Subsection 7.2.4, identifying the most promising features, which are then passed to a *feature-based classification*. In Subsection 7.2.5, we present how a classifier is estimated using these features and how it infers the class label of a time series.

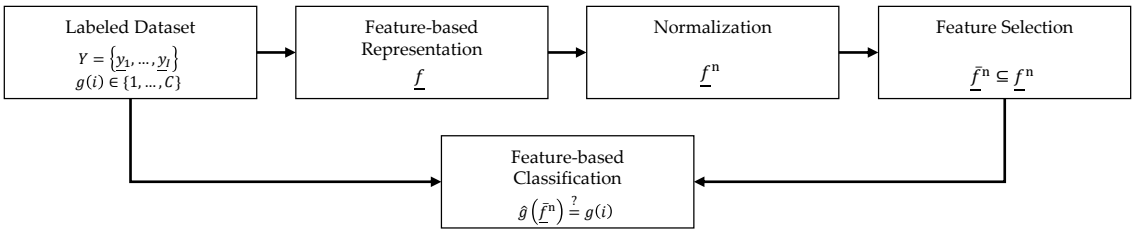


Figure 7.3: Overview of Feature-based Classification

7.2.1 Labeled Dataset

In contrast to the time series generation, matching, and clustering, time series classification relies on a labeled dataset, i.e., each time series is annotated by a class label. Recall that $\underline{y}_i$ is the i-th time series of a dataset Y. Formally, a label function g maps a time series to a label c from a set $\{1, \ldots, C\}$:

$$g(i) = c \tag{7.1}$$

For example, the Metering dataset contains three labels $C = 3$, classifying households ($c = 1$), small or medium businesses ($c = 2$), as well as other entities ($c = 3$). Each time series has a class label, even if the class is not well characterized, such as $c = 3$.

7.2.2 Feature-based Representation

We transform the labeled dataset using our feature-based representation (Chapter 4). First, each time series is treated as a composition of components:

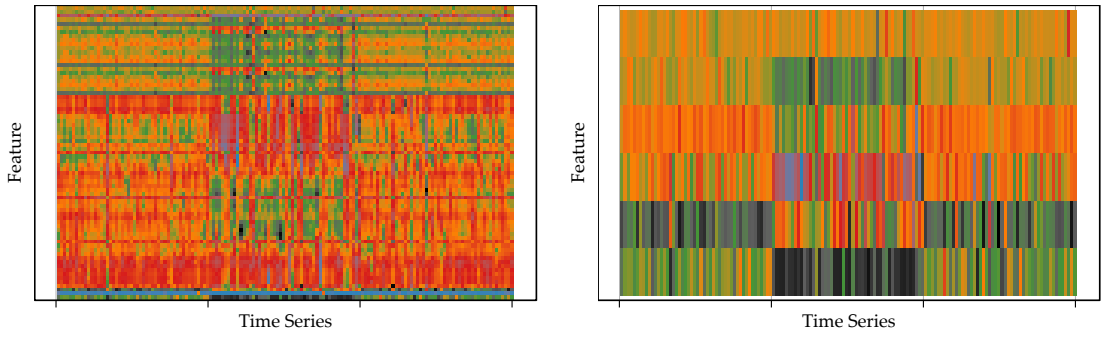
$$\underline{y} = \underline{tr} + \sum_{s=1}^{S} \underline{seas}_s + \underline{res} \tag{7.2}$$

where $\underline{tr}$, $\underline{seas}_s$, and $\underline{res}$ are the trend, seasonal, and residual component, respectively. Second, the components are reduced to deterministic and stochastic features:

$$\underline{f}^{\intercal} = (\theta_1(\underline{tr}), \theta_2(\underline{tr}), \sigma_1(\underline{seas}_1), \ldots, \sigma_{L_S}(\underline{seas}_S), sd(\underline{res}), skew(\underline{res}), kurt(\underline{res}), acf_1(\underline{res})) \tag{7.3}$$

where θ_1, θ_2, σ_l are deterministic features, representing the base value, the trend slope, and the season masks of the time series, respectively; and sd, $skew$, $kurt$, and acf_1 are the stochastic features, representing the standard deviation, the skewness, the kurtosis, and the autocorrelation of lag 1 of the residuals, respectively.

Figure 7.4 illustrates the features of the Metering dataset as a heatmap. This dataset contains time series from three classes, and we draw 50 time series from each class and arrange them next to each other on the x-axis. We arrange all features of Equation 7.3 on the y-axis and map the feature value to a color (Figure 7.4a). The first class and the second class are different to each other which illustrate the different feature values. The third class is highly similar to the first class.



(a) All Features (b) Selected Features

Figure 7.4: Feature-based Representation of Metering Dataset as Heatmap

7.2.3 Normalization

Different features exhibit different value ranges. For example, the autocorrelation ranges between -1 and $+1$, while the standard deviation is always positive. To avoid a bias in favor of features with larger ranges, we normalize features to a common range with a normalization technique. Let f^n denote a normalized feature. We choose among three different normalization techniques:

0-1-Normalization The 0-1-normalization transforms feature values to the interval $[0, 1]$. Recall that $F = \{f(\underline{y}_i), 1 \leq i \leq I\}$ are the values that a feature exhibits in a dataset. Formally, the 0-1-normalization is:

$$f^{01}(\underline{y}) = (f(\underline{y}) - min(F))/(max(F) - min(F)) \tag{7.4}$$

where $min(F)$ and $max(F)$ are the minimum and the maximum value of the feature, respectively.

0-1-Normalization with Outliers The 0-1-normalization with outliers is the normalization as presented in Section 4.5 (page 49). Recall that $IQR(F) = Q_3(F) - Q_1(F)$ is the interquartile range between the lower quartile $Q_1(F)$ and the upper quartile $Q_3(F)$. Using $lower(F)$ and $upper(F)$:

$$\begin{aligned} lower(F) &= \arg\min_{f(\underline{y}) \in F} \{f(\underline{y}) \geq median(F) - 1.5 \cdot IQR(F)\} \\ upper(F) &= \arg\max_{f(\underline{y}) \in F} \{f(\underline{y}) \leq median(F) + 1.5 \cdot IQR(F)\} \end{aligned} \tag{7.5}$$

the 0-1-normalization with outliers is defined as follows:

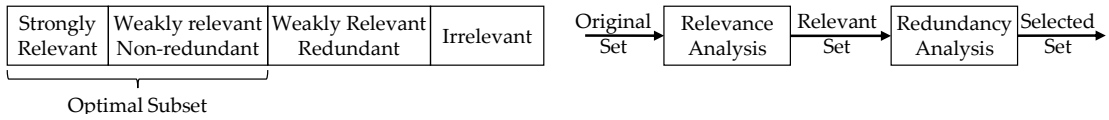
$$f^{01out}(\underline{y}) = (f(\underline{y}) - lower(F))/(upper(F) - lower(F)) \tag{7.6}$$

It transforms the feature value to a range between 0 and 1 except for extreme outliers.

Z-Normalization Similar to the z-normalization of time series, the z-normalization of a feature transforms its values to a range such that their mean is zero and their standard deviation is one:

$$f^z(\underline{y}) = (f(\underline{y}) - mean(F))/sd(F) \tag{7.7}$$

where $mean(F)$ an $sd(F)$ are the mean and standard deviation of the feature values, respectively.



(a) Subsets of Features (b) Workflow

Figure 7.5: Correlation-based Feature Selection [LL18]

While one eager classifier suggests a z-normalization [MDH+19], we cannot know how normalization will impact other classifiers. Therefore, we compare all three normalization techniques during a configuration of the engineering techniques.

7.2.4 Feature Selection

Our feature-based representation contains tens to hundreds of features, depending on the number and length of the season masks (Equation 7.3). Classification relies only on features with a high discriminative power, which is why these features should be selected prior to classifier estimation. *Feature selection* carries out this task with the overall goal to provide a comprehensive feature subset [LL18]. It filters *irrelevant* features with little discriminative information as well as *redundant* features whose discriminative information is largely covered by already selected features. This subset is beneficial for the subsequent classification: Not only does it reduce the estimation runtime, it also improves the classification accuracy.

The *correlation-based* feature selection is an important class of feature selection techniques (Figure 7.5) [LL18]. It focuses on the correlation among features, and the correlation between features and classes. The former is regarded as redundancy, the latter as relevance. Thus, a feature vector is divided into four subsets: (1) strongly relevant features, (2) weakly relevant but non-redundant features, (3) redundant features, and (4) irrelevant features (Figure 7.5a). Optimally, only relevant and non-redundant features should be selected. The correlation-based feature selection consists of two steps (Figure 7.5b). The *relevance analysis* determines the relevant features. Subsequently, the *redundancy analysis* determines and eliminates the redundant features and lead to the optimal subset of features.

Initially, a feature vector on Metering contains 73 features (Figure 7.4a). After the correlation-based feature selection, six features are selected on the Metering dataset (Figure 7.4b). They are the standard deviation, the autocorrelation of lag 1, the trend slope, and three seasonal features from the daily and weekly seasonal component.

Besides correlation-based feature selection, there is a second class of techniques, *search-based* feature selection. However, correlation-based feature selection has several advantages regarding efficiency and effectiveness over this second class [LL18], which is why we adopt it for our approach.

7.2.5 Feature-based Classification

Based on the normalized and selected features $\underline{\bar{f}}^n$ of the training dataset, a classifier $\hat{g}$ is estimated, which maps the features to a class label:

$$\hat{g}(\underline{\bar{f}}^n) = c \tag{7.8}$$

Since we reduced the time series to a low-dimensional feature-based representation, we can apply standard classification techniques to carry out the data-mining task. We give more details on these standard techniques in the experimental evaluation, together with their configuration.

After estimation, the classifier is used on a test dataset to assess whether it correctly infers the class of an unlabeled time series, i.e., a time series whose label the classifier does not know:

$$\hat{g}(\underline{\bar{f}}^n) \stackrel{?}{=} g(i) \tag{7.9}$$

If the classifier accurately infers a class label, then the underlying engineering technique is highly discriminative, which is evaluated experimentally.

7.3 EXPERIMENTAL EVALUATION

We evaluate our feature-based representation and compare it to the state-of-the-art techniques regarding an effective and efficient classification. We give our experimental setting (Subsection 7.3.1) and present and discuss the results (Subsection 7.3.2).

7.3.1 Experimental Setting

This subsection details the experimental setting, i.e., the selected datasets, the applied engineering techniques for time series classification, the eager and lazy classification techniques, and the measured output variables.

The engineering and classification techniques are implemented in R, as they rely on several methods of this programming language. They are published together with the scripts of the evaluation [Keg19b]. The experiments are executed and optimized for parallel execution on a server machine with a Twelve-Core Intel(R) Xeon(R) Gold Processor 6136@3.0GHz and 64GB of RAM.

Engineering Techniques

We detail the engineering techniques applied for time series classification along with their configuration.

Raw-data-based Engineering (Raw) As a baseline technique, we leave the time series as is without any transformation. An unlabeled time series is mapped to the label of its nearest neighbor in the training dataset, with respect to the Euclidean distance. We implement this technique in C, using optimizations from [RCM+12]: (1) we omit the calculation of the square root of the Euclidean distance since it does not change the ordering of the neighbors, and (2) we prune unpromising candidates if their Euclidean distance exceeds the best-so-far Euclidean distance.

Run Length Distribution (RLD) We re-implement the run length distribution from Bagnall and Janacek, as they did not provide access to their code [BJ14]. We assume that values $y = 0$ belong to a run length above the x-axis. The representations are compared with dynamic time warping, as suggested by the authors. We use the implementation of the R *dtw* package [Gio19].

Discrete Wavelet Transform (DWT) We transform a time series into Haar wavelets using the R *wavelets* package and its default configuration [Ald19]. Feature selection is carried out by selecting the $k \in \{2, 4, 8, 16\}$ strongest wavelet coefficients across the dataset, as suggested by [Mör03]. This feature selection is called *Best* by the authors.

Large Feature Vector (tsfresh) We compare our feature-based representation against the large feature vector from the tsfresh implementation, which is publicly available [CBNKL18]. Calculating all 794 features is infeasible for our datasets for two reasons. First, the time series of the Metering dataset have a length $T = 35{,}040$. Therefore, the representation of one single time series with tsfresh needs on average 30 minutes, i.e., a single-threaded execution of the dataset $I = 6{,}089$ needs approximately 126 days. Second, some features require a temporary memory size of $2 \cdot T^2$, i.e., approximately $20\ Gb$ per time series which is why the representation cannot be carried out in parallel execution on our machine. We select a subset of 748 features that the authors annotate as comprehensive and efficient. Moreover, we use the feature selection technique proposed by the authors, which is based on hypothesis tests rather than correlation [CKLF16].

Feature-Based Representation (FBR) We assess our feature-based representation (Equation 7.3), which decomposes a time series into its seasonal and trend components, and reduces each deterministic component to its deterministic features (base value, trend slope, season mask) as well as the residual component to its stochastic features (standard deviation, skewness, kurtosis, autocorrelation of lag 1). The feature vector has a length $K = 6 + \sum_{s=1}^{S} L_s$, where S is the number of seasons and L_s is the season length. We carry out a correlation-based feature selection using the WEKA framework [FHW16].

Classification Techniques

Since we transformed the time series in a low-dimensional space, we can apply standard classification techniques to carry out the data-mining task. As eager techniques, we select a *decision tree*, a *support vector machine*, and a *gradient boosting machine*. As a lazy technique, we select a *one-nearest-neighbor classifier*.

Decision Tree (DT) As a baseline technique for eager classification, we use a decision tree classifier. It separates the observations by automatically selecting an appropriate feature for splitting the feature space along one feature dimension. As features are evaluated one after another, there is no need to normalize them to the same range. The *Gini coefficient* is applied to select an appropriate split feature automatically. If splits do not improve the fit they are pruned during cross-validation. We use the implementation of the R *rpart* package for this technique [TAR19].

Support Vector Machine (SVM) The second eager classifier is a support vector machine (SVM). It considers the features of a time series as a point in the feature space, and fits a model which splits the classes of points by a gap that is as wide as possible. In contrast to a decision tree, the split is not done for each feature separately but for all features together, often leading to a higher classification accuracy. Therefore, appropriate features need to be selected and normalized to the same range. If the points are not separable in the feature space, SVM maps them into a higher space using a *kernel function*, which often increases separability. We use the implementation of the R *e1071* package for this technique [MDH+19] and evaluate a linear and a radial basis kernel function.

Gradient Boosting Machine (GBM) Gradient boosting is an ensemble technique which evolves a set of weak classifiers adding up to a combined classifier with a high classification accuracy. We use decision trees as weak classifiers. Consequently, gradient boosting selects features automatically and does not need scaling. However, there are several metaparameters that are needed to fine-tune a model: the *learning rate* to prevent overfitting; the *maximum depth* of a decision tree: while deeper trees contain more discriminative information but are also likely to overfit; and the number of *boosting steps* that iteratively lead to the combined classifer. We use the implementation of the R *xgboost* package [CHB+19], validate several metaparameter configurations: a learning rate $\{0.2, 0.3, 0.4\}$, a maximum depth $\{2, 4, 6, 8\}$, and a maximum number of boosting steps of $10{,}000$. We stop boosting if the accuracy does not improve within 100 steps.

One Nearest Neighbor (1NN) The one nearest neighbor is a lazy classifier. It maps a time series to the label of its nearest neighbor in the dataset. We carry out this classification on all engineering techniques, including the raw-data-based engineering. We calculate the Euclidean distance to compare the representations except for the run length distribution, which requires dynamic time warping [BJ14].

Datasets

We use two real-world datasets from the energy and the economic domain. We split the data into a training dataset of 80% for classifier estimation and validation and into a test dataset of 20% for using the classifier. Table 7.3 summarizes the dataset properties.

Metering (2 classes) We apply the Metering dataset as presented in Section 4.1 (page 42). In order to extract the yearly season of, we assume that the months January 2011 until June 2011 behave like January 2010 until June 2010. Thus, the dataset contains complete time series with 35,040 time instances. Besides the yearly season, all time series exhibit daily and weekly seasonal components. There are three classes: households ($c = 1$), small or medium businesses ($c = 2$), as well as other entities ($c = 3$). As we see in Figure 7.4, time series from class 3 are very similar to class 1; they are barely classifiable and thus, only disturb our evaluation. Therefore, we only focus on the 4,710 time series from class 1 and 2.

Payment The second dataset is taken from the IJCAI-2017 Data Mining Contest [Int17]. It consists of payment transactions of 2,000 distinct shops in hourly granularity monitored over 494 days. The shops are classified into food stores ($c = 1$), supermarkets ($c = 2$), entertainment stores ($c = 3$), health stores ($c = 4$), beauty stores ($c = 5$), and other shopping entities ($c = 6$). There are mainly time series from food stores and shopping entities, the other classes are sparse. None of the time series is complete; either a time series is not monitored from the beginning or it has missing values. In compliance with our definition of a time series dataset, we make this dataset complete by setting missing values to 0, i.e., no payments have been carried out in this hour. The dataset exhibits a daily and a weekly season which we use for the evaluation.

Output Variables

We assess the effectiveness and the efficiency of the engineering techniques with two output variables: the *classification accuracy* and the *overall runtime*.

The classification accuracy is defined as the share of labels that is correctly infered by the classifier. Recall that $g(i)$ is the correct label of the time series $\underline{y}_i$. Let $\hat{g}(\underline{\bar{f}}^n)$ be the label infered by the classifier. Then, the classification accuracy acc is defined as:

$$acc = \frac{1}{U} \sum_{c=1}^{C} \sum_{i:g(i)=c} I(g(i) = \hat{g}(\underline{\bar{f}}^n)) \tag{7.10}$$

where U is the size of the test dataset, C is the nuber of classes, and $I(\cdot)$ is the indicator function which returns 1 if the classes match and 0 otherwise.

The efficiency of engineering techniques is evaluated with the overall runtime, which encompasses the runtime for each step of our system (Figure 7.3), i.e., the *representation*, *normalization*, *feature selection*, and *classification runtime*. The classification runtime is further split into the runtime for *estimating* the classifier, and *using* the classifier. We measure the runtime as wall-clock time in seconds.

7.3.2 Results and Discussion

The results of our evaluation are presented and discussed in three parts. We start by choosing the best configuration for each engineering technique and for each dataset. Based on this configuration, we assess the effectiveness of each engineering technique in terms of classification accuracy, using the best eager and lazy classifier. Finally, we also compare their efficiency by measuring the overall runtime for classifying time series.

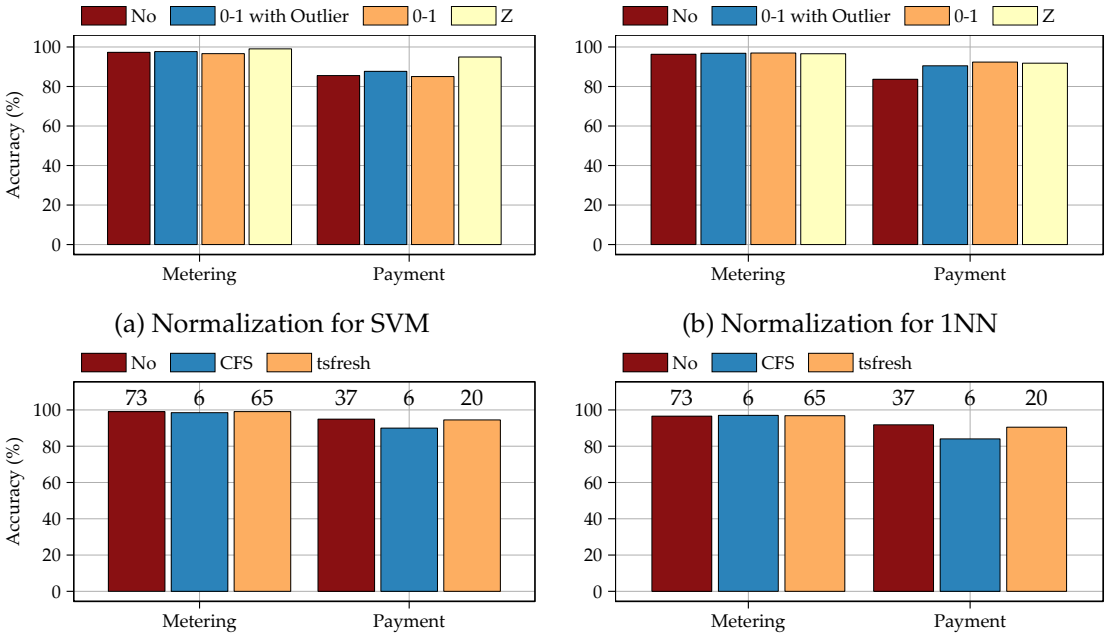
We choose the best configuration for an engineering technique and classification technique based on a ten fold cross-validation on the training dataset. Thus, results from the configuration of engineering techniques refer to the classification accuracy on the training dataset, while later results refer to the classification accuracy on the test dataset.

Configuration

The representation techniques FBR, tsfresh, and DWT are configured regarding normalization and feature selection. RLD does not require these steps, the representation is used as is [BJ14]. We select the configuration with the highest classification accuracy on the training dataset.

Table 7.3: Datasets for Classification

	Dataset Size	**Series Length**	**Frequency of Class Labels**	
			Training Dataset	**Test Dataset**
Metering (2 classes)	4,710	35,040	[3,380; 388]	[845; 97]
Payment (6 classes)	2,000	11,856	[1,132; 0; 1; 1; 463]	[283; 1; 1; 1; 1; 116]



(a) Normalization for SVM

(b) Normalization for 1NN

(c) Feature Selection for SVM

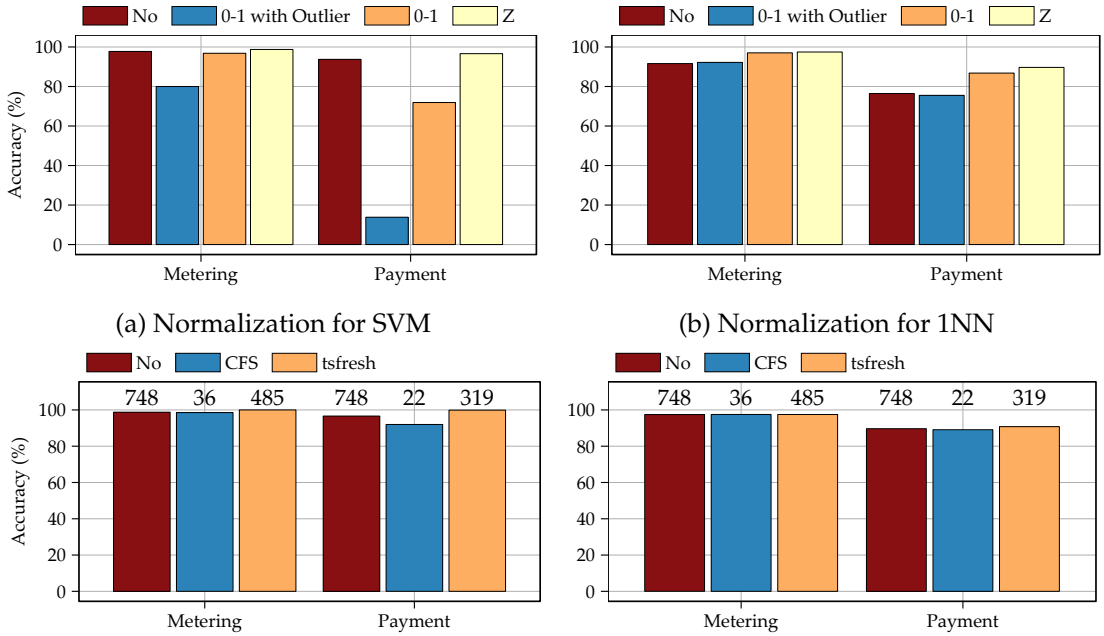
(d) Feature Selection for 1NN

Figure 7.6: Configuration of FBR

FBR Figure 7.6 visualizes the accuracy of FBR on the Metering and Payment dataset. First, it compares FBR for an SVM classifier using different normalizations (Figure 7.6a). On both datasets, the accuracy of an SVM classifier is highest using a z-normalization (cream white bar), which is typical for SVM [MDH+19]. 0-1-normalization (orange bar) and 0-1-normalization with outliers (blue bar) range at the same level as no normalization (red bar). The 0-1-normalization leads to the highest accuracy of an 1NN classifier (Figure 7.6b). While this gain is less significant on the Metering dataset, it is more significant on the Payment dataset. Thus, we focus on 0-1-normalized features when using FBR for the 1NN classifier.

Moreover, we assess whether feature selection has to be carried out for FBR, and compare the SVM classification accuracy (Figure 7.6c), using no feature selection (red bar), correlation-based feature selection (blue bar), and tsfresh feature selection (orange bar). The values above the bars indicate the length K of the feature vector. On Metering, the feature vector can be reduced by CFS and tsfresh without loosing much accuracy. There are several redundant and irrelevant features that are not needed for classifying this dataset, which is mainly due to several long season masks containing highly correlated values. However, on Payment, feature selection leads to a significant accuracy loss compared to the original feature vector, which we would like to avoid. 1NN reconfirms this behavior on both datasets (Figure 7.6d). Therefore, we focus on the 6 selected features on the Metering dataset, and on the 37 FBR features on Payment.

tsfresh The competitor tsfresh also needs a suitable configuration for SVM and 1NN regarding normalization and feature selection (Figure 7.7). For a SVM classifier, z-normalization leads again to most accurate results (Figure 7.7a). Interestingly, it is also the best normalization for a 1NN classifier (Figure 7.7b), which is why we use z-normalization for both classifiers. 0-1-normalization with outliers performs poorly because its value range for extreme outliers exceeds the usual value range for SVM.



(a) Normalization for SVM

(b) Normalization for 1NN

(c) Feature Selection for SVM

(d) Feature Selection for 1NN

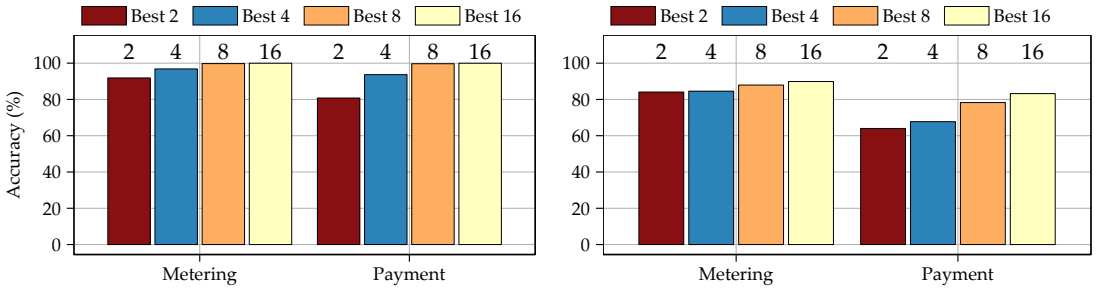
Figure 7.7: Configuration of tsfresh

Regarding feature selection, the technique from tsfresh (orange bar) provides the highest accuracy on both datasets and on both classifiers, SVM and 1NN (Figures 7.7c and 7.7d). No feature selection (red bar) is slightly (Metering) or even significantly (Payment) less accurate than tsfresh. Although CFS (blue bar) strongly reduces the representation size it also decreases the accuracy, apparently loosing some relevant features. Therefore, we focus on the tsfresh feature selection for SVM and 1NN.

DWT The second competitor, DWT, does not use normalization, but it has to select the most discriminative features from the wavelet transform. We use the feature selection Best and assess feature vectors of length 2, 4, 8, and 16, as suggested by Mörchen [Mör03]. Using these lengths, Figure 7.8 visualizes the classification accuracy of SVM and 1NN (Figures 7.8a and 7.8b). Both classifiers provide most accurate results for the largest feature vector, Best 16 (cream white bar), Best 2, Best 4, and Best 8 decrease the accuracy. Therefore, we apply Best 16 for SVM and 1NN. Although DT and GBM select features automatically, presenting the DWT feature vector without selection is infeasible for these techniques, as this vector still have the same length as the time series. Therefore, we present only the 16 best features to these classifiers, too, which is the most accurate feature selection according to Figure 7.8c and 7.8d.

Classification Accuracy

After the configuration of the techniques, we now select the most accurate eager and lazy classifier for each technique based on the training dataset. This classifier is selected and assessed on the test dataset (Figure 7.9). For DWT, tsfresh, and FBR, we display the accuracies of the best eager classifier (Figure 7.9a), indicating the vector length of the selected features above the bar. DWT (cream white bar) with its best configuration achieves 89.7% on Metering and 86.4% on Payment, while tsfresh (orange bar) achieves



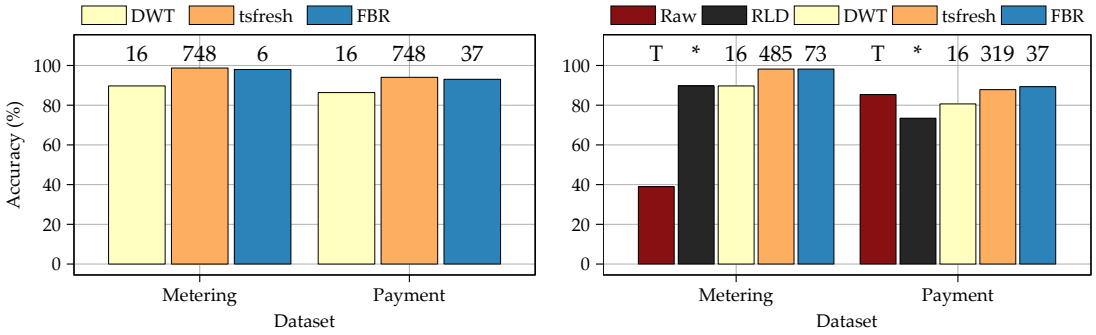
(a) Feature Selection for SVM

(b) Feature Selection for 1NN

(c) Feature Selection for DT

(d) Feature Selection for GBM

Figure 7.8: Configuration of DWT



(a) Eager Classifiers

(b) 1NN Classifier

Figure 7.9: Classification Accuracy

up to 98.7% and 94.0%, respectively. Our technique, FBR (blue bar), achieves almost the same results than tsfresh, 98.0% and 93.1%, respectively. Importantly, it achieves these results with a significantly smaller feature vector, using 6 features for Metering and 37 features for Payment. In contrast, tsfresh requires 748 features on both datasets. Let us now turn to representation accuracy of the 1NN classifier (Figure 7.9b). Again, the vector length is indicated above the bars. The raw-data-based technique (red bar), using the full time series of length T as input, has an accuracy of 39.0% and 85.4% on Metering and Payment, respectively. The poor result on Metering emphasizes the need to transform a time series into a representation before discriminative features may be discovered. The technique RLD (gray bar) and DWT (cream white bar) achieve both 89.7% on Metering, as well as 73.4% and 80.6% on Payment. Feature vectors from tsfresh (orange bar) achieves 98.2% on Metering and 87.8% on Payment. The most striking observation to emerge from this comparison is that FBR (blue bar) has the same result as tsfresh on Metering, but outperforms tsfresh on Payment, having 89.3%. Using a lazy 1NN classifier, FBR achieves better results than tsfresh while having a much smaller feature vector.

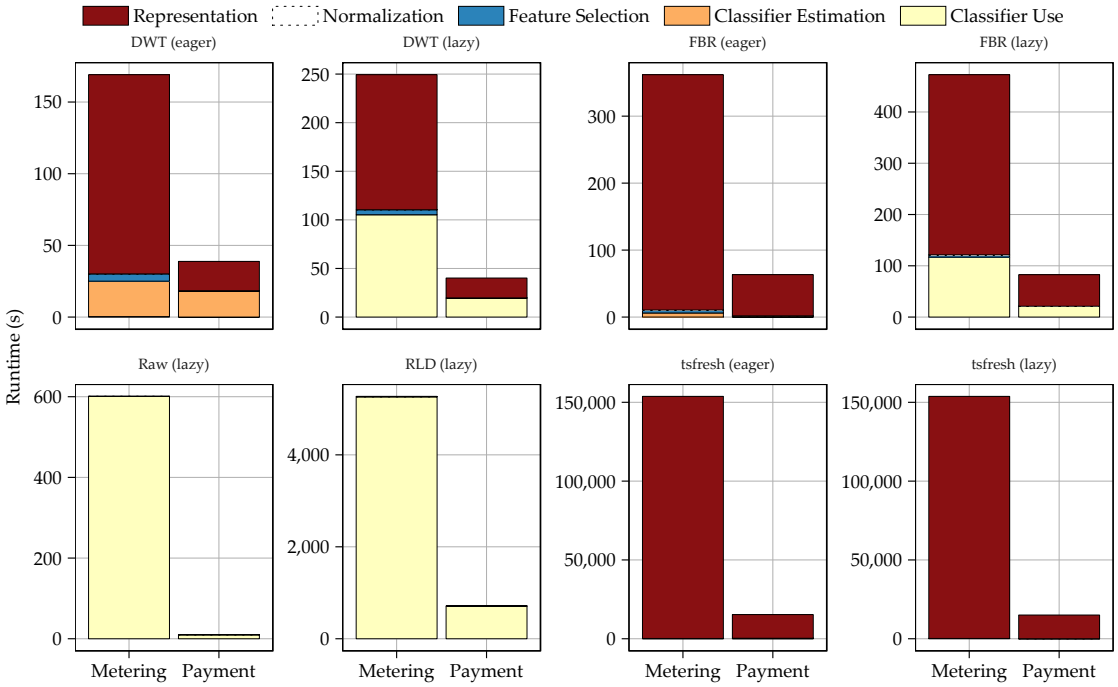


Figure 7.10: Overall Runtime

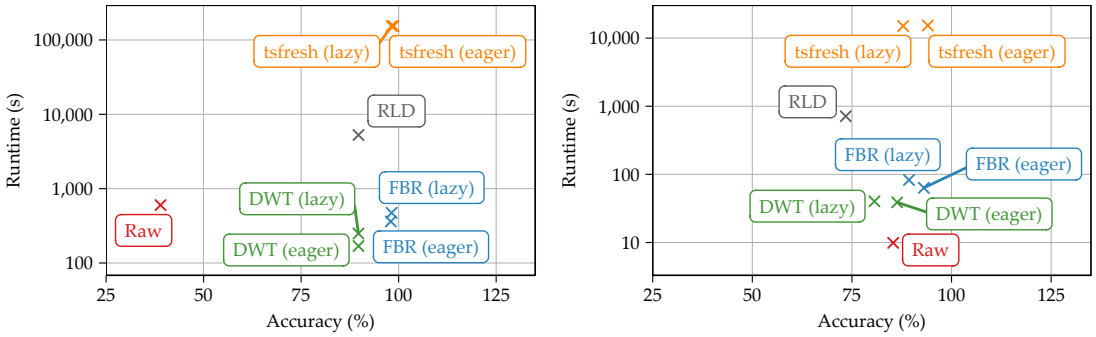
Using eager and lazy classifiers, FBR provides results which are as accurate as large feature vectors, or even more accurate. Besides, FBR achieves this with a much shorter feature vector indicating that it captures highly discriminative features.

Overall Runtime

Figure 7.10 presents the overall runtime in seconds of the competitors in increasing order. It includes the representation runtime (red bar), the feature selection runtime (blue bar), as well as the classification runtime of the most effective classifier on the test dataset. The classification runtime is split into estimation and use runtime (orange and cream white bar). The normalization runtime is also included in the overall runtime. However, it is negligible compared to the other runtimes and thus, it is not visible.

DWT, FBR, and tsfresh spend most of the time representing the time series. DWT is fastest, as it requires only one pass over the time series, while FBR requires more than one pass for the multi-seasonal decomposition (Algorithm 4.1, page 45). On Metering, these representations require approximately 30 and 75 milliseconds per time series, respectively. In contrast, tsfresh is much slower, requiring approximately 30 seconds per time series. All three techniques are implemented in R or Python with underlying C code, but tsfresh calculates many features and thus, requires more passes over a time series. A lazy 1NN classifier takes more classification runtime than an early classifier. If present, the feature selection has a negligeable runtime compared to representation and classification. The representation runtime of Raw is zero, and that of RLD is negligible. These techniques spend most of the time using the 1NN classifier, i.e., calculating the distances to all labeled time series from the dataset and selecting the label from the less distant time series. The raw-data-based technique uses the Euclidean distance, which is fast compared to the DTW distance measure applied to RLD.

Taken together, these findings suggest that FBR is among the most efficient classifiers for classifying long time series.



(a) Metering Dataset

(b) Payment Dataset

Figure 7.11: Classification Accuracy versus Overall Runtime

Accuracy and Runtime

Figure 7.11 displays a scatterplot of the classification accuracy (x-axis) and the overall runtime (y-axis), using a logarithmic y-axis for a better visualization. It is desirable that a technique has a high accuracy, i.e., it is the rightmost compared to the competitors, and it has a short runtime, i.e., it is lower than the competitors. On Metering, FBR ranges in the lower right corner of the plot, i.e., it provides a trade-off compared to its competitors tsfresh, which is slower but slightly more accurate, and DWT, which is faster but less accurate. This trade-off is also confirmed on the Payment dataset, where the raw-data-based 1NN classifier becomes the fastest but less accurate competitor. This comparison reconfirms the aforementioned observations: FBR provides very accurate classification results, while keeping a short overall runtime.

7.4 SUMMARY

In summary, our work emphasizes that big time series datasets consisting of long time series require not only effective but also efficient engineering techniques for classification. Regarding effectiveness, our evaluation suggests that feature-based engineering techniques outperform raw-data-based and shape-based techniques. In particular, our feature-based engineering technique classifies time series as accurately as the best competitor, but it requires significantly fewer features. It is quite natural that feature-based techniques spend most of the time representing the dataset, while raw-data-based and shape-based techniques spend most of the time using the 1NN classifier. Regarding the overall runtime, feature-based techniques outperform other techniques if they transform a dataset efficiently. In particular, our feature-based engineering technique is as fast as the fastest competitor, yet it provides more accurate results. Thus, our evaluation stresses the importance of a trade-off regarding these two objectives. Our feature-based engineering technique classifies as accurately as tsfresh, but it is orders of magnitude faster.

These observations have two main implications for research into time series analytics. First, the classification of big time series datasets is a new challenge that researchers should focus on with benchmark datasets and efficient engineering techniques. Second, the 1NN classifier with the Euclidean distance measure is a poor baseline technique for big time series datasets. In time series matching (Chapter 6), this distance measure is considered the reference for subsequent approximations (SAX, sSAX), for efficiently retrieving an exact match. However, as our evaluation suggests, an exact match provides inaccurate labels, which is why other baseline techniques for both, matching and classification, are desirable for big time series datasets.

8

TIME SERIES CLUSTERING

TIME SERIES FORECASTING is a well-established research subject and an essential part of today's business landscape in many domains. The load control in the energy domain and decision-making in the economic domain are just two examples where forecasting is of paramount importance. While the quality of time series forecasting, i.e., the *forecast accuracy*, is still most important, the computation has to keep up with the challenges of big time series datasets.

A *forecast technique* identifies and estimates a *forecast model* for a given time series. Subsequently, this model infers a *forecast* of the future time series values. While *traditional* forecast techniques provide one forecast model per time series, *vectorized* techniques provide one model per dataset, and reduce the runtime for model estimation. Recently, Hartmann et al. proposed the *cross-sectional autoregression model (CSAR)* [Har18]. This vectorized forecast technique assumes that time series of a dataset have a similar autoregression and misprediction behavior. For example, the Metering dataset contains thousands of time series which are inherently similar because all of them capture electricity consumption, with daily, weekly, and yearly seasons. Thus, model estimation provides one *monolithic* CSAR model per dataset.

However, these assumptions only hold to a certain degree. As mentioned by Hartmann et al. [HRH+19], clustering time series by their behavior is expected to increase the forecast accuracy. Moreover, our heatmap of features revealed that the households exhibit other features than small and medium businesses (Figure 7.4). The underlying consumers of the aforementioned Metering dataset are different, they are households, small or medium businesses, as well as other entities. In this chapter, we aim to identify CSAR models for *clusters* of time series. They may increase the forecast accuracy but also maintain a short runtime, as there are only a few models to compute.

Time series clustering partitions a time series dataset into homogeneous clusters such that the distance between clusters is maximal. It is often applied as subroutine of other data-mining tasks, which is why we apply it to time series forecasting [ASY15]. Research focuses on engineering for time series clustering as well as the selection of clustering techniques and clustering prototypes [ASY15].

As we described in Chapter 3, feature-based engineering tackles the challenges of big time series datasets very well. Therefore, we apply a feature-based clustering to partition a time series dataset into clusters and forecast each cluster with an individual CSAR model. The evaluation of this approach shows encouraging results as it significantly improves the forecast accuracy while maintaining a shorter runtime time than traditional forecast techniques.

In detail, we give an overview of cross-sectional forecasting with CSAR in Section 8.1 and highlight its ability to address the challenges of big time series datasets. Subsequently, in Section 8.2, we present a feature-based clustering technique and use it to partition time series into clusters. Each cluster is estimated and forecasted with an individual CSAR model. In Section 8.3, we conduct an experimental study to show that CSAR models on time series clusters improve the forecast accuracy significantly compared to a monolithic CSAR model, while maintaining a short runtime. Section 8.4 summarizes the results.

8.1 CROSS-SECTIONAL AUTOREGRESSION MODEL

The cross-sectional autoregressive (CSAR) model is inspired by the ARIMA model, as it utilizes the autoregression and the error terms of the data. In contrast to ARIMA, CSAR is a vectorized forecast technique, which assumes that time series in a dataset are similar. It identifies and estimates one model per dataset, which is then used to provide a forecast for each time series. The core concept of CSAR is the *cross section* which gathers values of all time series from a dataset measured at one time instance. Formally, it is defined as:

$$\underline{y}_t^\intercal = (y_{t,1}, \ldots, y_{t,i}, \ldots, y_{t,I}) \text{ where } \underline{y}_t \in \mathbb{R}^I, 1 \leq t \leq T \tag{8.1}$$

where $y_{t,i}$ is the value of the i-the time series at time instance t. Thus, it shifts from a horizontal view, where time series are considered individually, to a vertical view, where time series are viewed commonly per time instance. Based on cross sections, CSAR estimates a model. A forecast for an cross-sectional autoregressive model of order 1 is of the form:

$$\hat{\underline{y}}_{t+1} = \hat{\mu} + \hat{\phi}_1 \underline{y}_t \tag{8.2}$$

where $\hat{\mu}$ is the mean and $\hat{\phi}_1$ is the weight, which are estimated beforehand.

CSAR models tackle the challenges of big time series datasets by providing one model per dataset, while traditional forecast techniques would estimate thousands of individual models. Subsequently, we re-visit its three components: integration, autoregression, and error terms. While the integration component is a prerequisite to make a time series stationary, autoregression and error terms are part of a CSAR model.

8.1.1 Integration

Integration is carried out to make a time series stationary by eliminating deterministic components. If integration is used, all time series of a dataset are differentiated before model identification and estimation. After model use, results are transformed back using integration.

For trend elimination, CSAR uses a differentiation similar to ARIMA (Equation 3.30, page 33):

$$v_t = (1 - B)^d y_t \tag{8.3}$$

where B is the backshift operator and v_t is the differentiated time series. For $d = 1$, this operation results in a differentiated time series $v_t = y_t - y_{t-1}$.

For season elimination, CSAR uses a differentiation taking into account the season length L (Equation 3.31, page 33):

$$v_t = (1 - B^L)^D y_t \tag{8.4}$$

For $D = 1$, this operation results in a differentiated time series $v_t = y_t - y_{t-L}$.

8.1.2 Autoregression

The autoregression component extends the AR process from ARIMA with cross sections. It is closely related to the AR component from ARIMA models, as it regards a time series value as a finite, linear, weighted aggregate of its previous values. For the non-seasonal and seasonal part, it uses p and P previous values, respectively. The corresponding weights ϕ_1, ϕ_2, ..., ϕ_p and Φ_1, Φ_2, ..., Φ_P as well as the mean value are estimated. Equation 8.5 shows the non-seasonal autoregression process. In contrast to the autoregressive process from ARIMA (Equation 3.21, page 32), cross sections are used instead of time series values:

$$\underline{y}_t = \mu + \phi_1 \underline{y}_{t-1} + \phi_2 \underline{y}_{t-2} + ... + \phi_p \underline{y}_{t-p} + a_t \tag{8.5}$$

Model parameters are estimated accordingly so that the model can be used for forecasting. A non-seasonal model without seasonal part provides a forecast as shown in Equation 8.6. A seasonal model without a non-seasonal part provides a forecast as shown in Equation 8.7. Both formulations rely on the estimated mean and weights.

$$\underline{\hat{y}}_t = \hat{\mu} + \hat{\phi}_1 \cdot \underline{y}_{t-1} + \hat{\phi}_2 \cdot \underline{y}_{t-2} + ... + \hat{\phi}_p \cdot \underline{y}_{t-p} \tag{8.6}$$

$$\underline{\hat{y}}_t = \hat{\mu} + \hat{\Phi}_1 \cdot \underline{y}_{t-L} + \hat{\Phi}_2 \cdot \underline{y}_{t-2 \cdot L} + ... + \hat{\Phi}_P \cdot \underline{y}_{t-P \cdot L} \tag{8.7}$$

An autoregressive process containing a seasonal and a non-seasonal part together requires a more general formulation than Equation 8.5. Recall that a seasonal AR process is given by $\Phi(B^s) y_t = \alpha_t$ (Equation 3.32) and that the error terms α_t are modeled by a second non-seasonal AR process given by $\phi(B)\alpha_t = a_t$ (Equation 3.33). Applied to cross sections, these two equations yield the general formulation of a cross-sectional autoregressive process:

$$a_t = \phi(B)\Phi(B^s) \cdot \underline{y}_t \tag{8.8}$$

For a process with $P = 1$ and $p = 2$, this process results in:

$$\begin{aligned} a_t &= \phi(B)\Phi(B^L) \cdot \underline{y}_t \\ &= (1 - \phi_1 B - \phi_2 B) \cdot (1 - \Phi B^L) \cdot \underline{y}_t \\ &= (1 - \phi_1 B - \phi_2 B) \cdot (\underline{y}_t - \underline{y}_{t-L}) \\ &= \underline{y}_t - \Phi_1 \cdot \underline{y}_{t-L} - \phi_1 \cdot \underline{y}_{t-1} - \phi_2 \cdot \underline{y}_{t-2} + \Phi_1 \phi_1 \cdot \underline{y}_{t-L-1} + \Phi_1 \phi_2 \cdot \underline{y}_{t-L-2} \end{aligned} \tag{8.9}$$

CSAR estimates a model for this process and also for the mean [Har18]:

$$\underline{\hat{y}}_t = \hat{\mu} + \hat{\Phi}_1 \cdot \underline{y}_{t-L} + \hat{\phi}_1 \cdot \underline{y}_{t-1} + \hat{\phi}_2 \cdot \underline{y}_{t-2} - \hat{\Phi}_1 \hat{\phi}_1 \cdot \underline{y}_{t-L-1} - \hat{\Phi}_1 \hat{\phi}_2 \cdot \underline{y}_{t-L-2} \tag{8.10}$$

8.1.3 Error Terms

Individual time series in a dataset may exhibit systematic misprediction when they are forecasted by a cross-sectional autoregressive model. To compensate for this behavior, CSAR uses error terms and applies them to correct the forecast. An error term e occurs while estimating a cross-sectional autoregressive model; it expresses the deviation of the forecast value from the real value in the training interval. CSAR combines q non-seasonal and Q seasonal error terms for correcting the forecast of time series $\underline{y}_i$:

$$e_t = y_t - \hat{y}_t \tag{8.11}$$

$$\bar{e}_{i,t} = \frac{1}{q+Q}\Big(\sum_{j=1}^{q} e_{i,t-j} + \sum_{j=1}^{Q} e_{i,t-j \cdot L}\Big) \tag{8.12}$$

$$\hat{y}_{i,t} = \hat{\mu} - \bar{e}_{i,t} \tag{8.13}$$

8.2 FEATURE-BASED CLUSTERING

CSAR assumes that all time series of a dataset exhibit the same behavior. In this section, we present a feature-based clustering technique efficiently identifying clusters for CSAR models. A CSAR model uses a set of metaparameters: p and P refer to the order of non-seasonal and seasonal cross sections, q and Q to the order of non-seasonal and seasonal error terms, and $constant$ to the availability of a non-zero mean value. It assumes that all the time series in a cluster exhibit the same behavior. They have the same degree of autoregression, expressed by a similar autoregressive process, and the same degree of systematic misprediction, expressed by similar error terms. Through the use of correlation features, we are able to partition a dataset with respect to this behavior. Originally, correlation features are used for ARIMA model identification. Therefore, we shortly recap the features based on this application. Second, we leverage them for partitioning datasets with CSAR.

8.2.1 ACF and PACF for ARIMA

ARIMA modeling needs an identification of metaparameters for each time series, using the autocorrelation and the partial autocorrelation function for the determination of the order of the moving average and the autoregressive process, respectively [BJR08].

The autocorrelation function (ACF) expresses the linear dependence of a time series on itself, shifted by a sequence of lags. For example, the ACF of lag 1 (Equation 3.45, page 37) expresses the linear dependence between a time series with itself lagged by one time instance. With ACF, ARIMA determines the order q of a moving average process. The first q ACF lags of a moving average process are nonzero, while subsequent lags are zero.

Another correlation feature, which has not been used as time series representation, is the *partial autocorrelation* (PACF). It also expresses the dependence of a time series on itself, shifted by a sequence of lags. In contrast to ACF, the dependence of time instances between the lag is removed. For example, $pacf_2$ expresses the dependence of y_t from y_{t-2} which is corrected by the effects of y_{t-1}. The partial autocorrelation $pacf_1$ expresses the dependence of y_t from y_{t-1} which is equal to acf_1 since there are no time instances between t and $t+1$. With PACF, ARIMA determines the order p of an autoregressive process. The first p PACF lags of an autoregressive process are nonzero, while subsequent lags are zero.

8.2.2 ACF and PACF for CSAR

ARIMA and CSAR are closely related. The autoregressive process of ARIMA is the blueprint for the autoregressive part of CSAR and the moving average process serves the same purpose as the error terms of CSAR. ACF and PACF advises ARIMA regarding these components. We hypothesize that these features are able to identify time series with similar behavior such these time series may be considered together in a cluster. Interestingly, ACF and PACF may be calculated efficiently for common lags $h \ll T$, which makes them suitable for big time series datasets.

Let p_{max}, P_{max}, q_{max}, and Q_{max} be the maximum number of non-seasonal and seasonal cross sections as well as non-seasonal and seasonal error terms that are assessed during

Table 8.1: Correlation Features for CSAR

Component	Features	Indices
Non-seasonal cross section	$pacf_k$	$1 \leq k \leq p_{max}$
Seasonal cross section	$pacf_{k \cdot s}$	$1 \leq k \leq P_{max}$
Non-seasonal error term	acf_k	$1 \leq k \leq q_{max}$
Seasonal error term	$acf_{k \cdot s}$	$1 \leq k \leq Q_{max}$

model validation. The correlation features as given in Table 8.1 build up a feature-based representation for neighboring time series with similar autoregression and misprediction.

The partitioning carried out as follows. The feature-based representation is reduced into two dimensions by t-distributed stochastic neighbor embedding (t-SNE) [vdMH18]. For each pair of embedded representations, the Euclidean distance is calculated and stored in a distance matrix. Based on this distance matrix, a density-based spatial clustering (DBSCAN) is carried out in order to discover partitions of the dataset [EKSX96]. With kNN-distplot, the DBSCAN parameters (minimum distance ϵ and minimum points $minPoints$) are appropriately configured.

8.3 EXPERIMENTAL EVALUATION

To examine whether feature-based clustering for CSAR improves the forecast accuracy, it is compared to monolithic CSAR and to commonly used forecast techniques. Moreover, its runtime for identifying metaparameters is assessed, as well as the runtime for model estimation and use.

8.3.1 Experimental Setting

This subsection details the experimental setting, i.e., the applied forecast techniques, the selected datasets, the measured output variables, and the configurations for representation, clustering, and forecast techniques.

To leverage R's built-in functions for optimization, clustering, and forecasting, CSAR and feature-based clustering for CSAR are implemented in R together with the scripts of the evaluation [R C18]. The experiments are executed and optimized for parallel execution on a server machine with a Twelve-Core Intel(R) Xeon(R) Gold Processor 6136@3.0GHz and 64GB of RAM.

Datasets

We use two real-world datasets, the Metering and the Payment dataset, and shortly summarize their characteristics. In compliance with Hartmann et al. [HRH+19], we also include time series which are not complete. ACF and PACF provide reasonable results for incomplete time series, thus, we can apply our feature-based representation. Moreover, we do not carry out integration on these datasets, as it did not lead to a systematic improvement in the forecast accuracy [HRH+19].

Metering (6 hours) We apply the Metering dataset presented in Section 4.1 (page 42). In compliance with Hartmann et al. [HRH+19], we aggregate the values to a 6-hour granularity. Thus, the dataset contains 6,433 time series with a length of 2,144. 5% of the data is missing. We evaluate our techniques on the weekly season, as done by Hartmann et al. [HRH+19].

Payment The second dataset is the Payment dataset as presented in Section 7.3 (page 100). We recall that it consists of payment transactions of 2,000 distinct shops in daily granularity monitored over 494 days. The shops are classified into food stores, supermarkets, entertainment stores, health stores, beauty stores, and other shopping entities. None of the time series is complete; either a time series is not monitored from the beginning or it has missing values. Compared to a complete dataset, 40% of the data is missing. The dataset exhibits a weekly season which we use for the evaluation.

Forecast Techniques

We compare the CSAR techniques with commonly used forecast techniques, i.e., two baseline techniques and one traditional forecast technique:

Naive (N) We include the naive forecast where a measured value is the forecast of the subsequent time instance: $\hat{y}_t = y_{t-1}$. This forecast reflects the baseline of our evaluation: if a forecast technique performs worse than the naive forecast, it is obviously not suited for the given dataset.

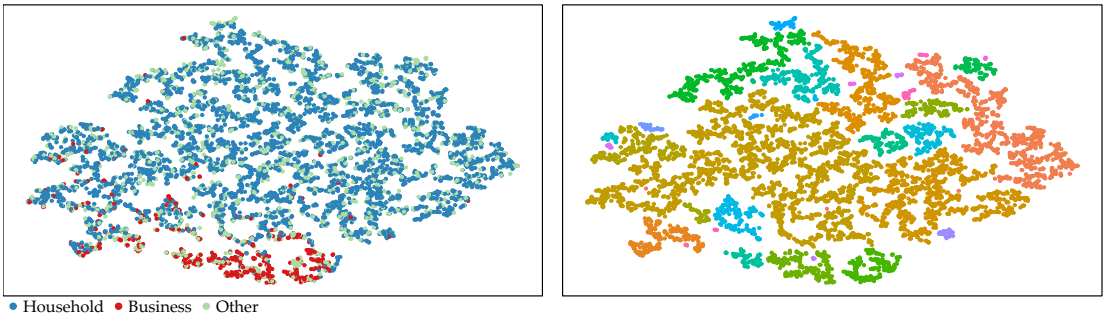
Naive Seasonal (NS) If a time series is seasonal, then a value is highly correlated to the value from the previous season. Based on the observation, we include the naive seasonal forecast as a second baseline: $\hat{y}_t = y_{t-s}$. Like the naive forecast, it should be a lower bound for subsequent forecast techniques.

ARIMA (A) As a traditional forecast technique, we select an ARIMA model as implemented in R. We use *auto.arima* from the *forecast* package [HK08] for model identification and *arima* from the *stats* package for model estimation and use. In compliance with CSAR, we set the maximum order to the same number of cross sections and error terms (p_{max} and q_{max}). A seasonal component is assumed with a maximum order in compliance with the seasonal cross sections and error terms of CSAR (P_{max} and Q_{max}). A mean value is assumed but not a trend. Missing values in a time series are handled by this implementation. If identification or estimation fails, the forecast for this time instance is missing.

CSAR with Single Cluster (OC) CSAR with "one global" cluster only trains a monolithic CSAR model for a full dataset, which provides a forecast for all time series simultaneously.

CSAR with Class-based Clusters (CC) The selected datasets provide class labels that we exploit for creating partitions. Each cluster identifies and models an individual CSAR model.

CSAR with Feature-based Clusters (FC) We use our feature-based representation and assume a maximum order of 2 for the ACF and PACF features. A DBSCAN clustering with $\epsilon = 1.5$ and $minPts = 3$ is applied. Each cluster identifies and models an individual CSAR model.



(a) Class-based (b) Feature-based

Figure 8.1: Clustering of Metering Dataset

Table 8.2: Metaparameter and Time Intervals

Dataset	Metaparameter Intervals					Time Intervals		
	p	P	q	Q	constant	Training	Validation	Use
Metering	[0, 2]	[0, 2]	[0, 2]	[0, 2]	[False, True]	[1, 2088]	[2089, 2116]	[2117, 2144]
Payment	[0, 2]	[0, 2]	[0, 2]	[0, 2]	[False, True]	[1, 438]	[439, 466]	[467, 494]

Figure 8.1 displays the feature-based representation in a two-dimensional feature space. The class-based clustering (Figure 8.1a) shows that households (blue points) and small- or medium businesses (red points) have a different behavior which is why they are not overlapping. However, the other entities (green points) are spread across the feature space. The feature-based clustering (Figure 8.1b) leads to 33 partitions plus one partition for outliers. On the Payment dataset, the feature-based clustering leads to 37 partitions plus one partition for outliers.

Model Validation and Use

Model validation is carried out for CSAR in order to determine the best metaparameters for each model. For all clusters, all metaparameter combinations, and all validation time instances (Table 8.2), a model is estimated returning a one-step ahead forecast for each time series in its cluster $\hat{y}_t, Tr + 1 \leq t \leq Tr + V$, where Tr and V are the number of time instances for model training and validation, respectively. The metaparameter combination providing the best forecast accuracy for its cluster is selected for model use. ARIMA selects the best metaparameters based on the recommendation of *auto.arima* on the training and validation interval $y_t, 1 \leq t \leq Tr + V$.

Model use is carried out to assess the forecast accuracy of all forecast techniques. For each time instance of the model use interval (Table 8.2), a model is estimated based on the selected metaparameter combination. The resulting vector of one-step ahead forecasts $\hat{y}_t, Tr + V + 1 \leq t \leq Tr + V + U$ is assessed regarding forecast accuracy, where U is the number of time instances for model use.

Output Variables

In compliance with Hartmann et al. [HRH+19], the forecast accuracy is assessed with the *symmetric absolute percentage error* (SAPE):

$$SAPE(y_t, \hat{y}_t) = \frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2} \cdot 100\% \tag{8.14}$$

where y_t is the real value and $\hat{y}_t$ the corresponding forecast value. SAPE is a relative lock-step distance measure which relates the error to the average of real and forecast value. Thus, it has a defined range (0% to 200%) and it is easier to interpret than *squared* lock-step distance measures such as the Euclidean distance or RMSE. If y_t or $\hat{y}_t$ are not defined, SAPE returns 200%. Averaging several SAPE errors results in the Symmetric Mean Absolute Percentage Error (SMAPE). While model validation applies this output variable on the validation interval $[Tr + 1, ..., Tr + V]$ to select the best metaparameters, model use applies it to the use interval $[Tr + V + 1, ..., Tr + V + U]$ to assess the forecast model:

$$SMAPE(\underline{y}, \underline{\hat{y}}) = \frac{1}{T} \sum\nolimits_t SAPE(y_t, \hat{y}_t) \tag{8.15}$$

The forecast accuracy is assessed on the base and top level. On the base level, every time series is assessed individually. On the top level, all time series are aggregated for each time instance: $\sum_i y_{t,i}$. These aggregated values are compared to the aggregated forecast $\sum_i \hat{y}_{t,i}$. If a real value is missing, it is not included in the aggregate together with its forecast counterpart (and vice-versa).

The runtime is measured in seconds. We only report the time of the single-threaded execution. Since ARIMA execution takes a considerable amount of time, we measure the calculation time for a representative sample of time series and extrapolate it for the complete dataset. The parallel execution, which is carried out on all time series for assessing the forecast accuracy, confirms the accuracy of this extrapolation.

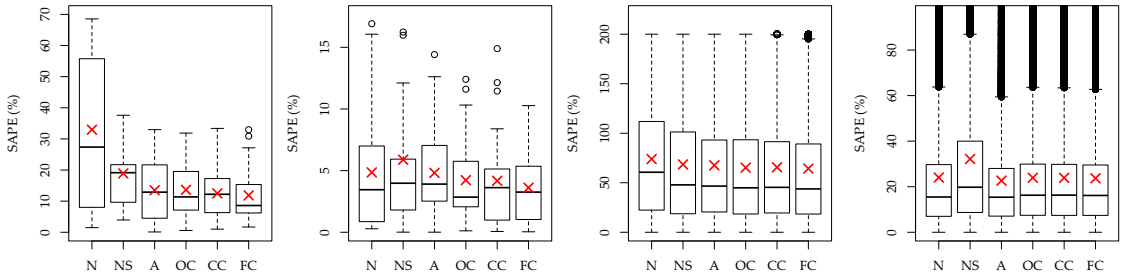
8.3.2 Results and Discussion

The results of our evaluation are presented and discussed in the order of our hypotheses: the forecast accuracy is assessed, followed by runtime for model validation and model use.

Forecast Accuracy

Figure 8.2 shows the forecast accuracy of the datasets on the top level and on the base level, respectively. The boxplot shows the distribution of forecast errors as SAPE, while the red cross displays the mean error as SMAPE.

On the top level of the Metering dataset (Figure 8.2a), the naive forecasts (N and NS) have an SMAPE of 32.9% and 18.9%. With 13.5%, ARIMA (A) is more accurate. The monolithic CSAR (OC) is as accurate as ARIMA with 13.6%. Clustering significantly improves the accuracy. Class-based clustering (CC) has an error of 12.5%. The feature-based clustering (FC) provides an even better partitioning which reaches 11.8%.



(a) Metering (Top) (b) Payment (Top) (c) Metering (Base) (d) Payment (Base)

Figure 8.2: Forecast Accuracy

This behavior is confirmed on the Payment dataset (Figure 8.2b). Again, the naive forecasts provide an SMAPE of 4.9% (N) and 5.9% (NS), respectively. Thus, the value of the previous season is not a good guess for this dataset and the most recent value provides a smaller error. ARIMA (A) is only slightly better with 4.8%. CSAR modeling revealed an overall improvement compared to these forecast techniques. Monolithic CSAR (OC) provides an error of 4.2%. There is no significant difference with respect to class-based clustering (CC); the latter also provides 4.2%. Interestingly, feature-based clustering (FC) provides a strong improvement with 3.6%, underlining that class labels do not necessarily reflect the behavior for accurate clustering.

Contrary to expectations, no significant difference is observed between the traditional and CSAR forecast techniques on the base level. The error on the Metering dataset (Figure 8.2c) decreases from 55.5% and 48.6% (N and NS) to 45.8% (A). OC and CC provide results that are slightly less accurate than ARIMA (46.8% and 47.2%). The feature-based clustering (FC) has an accuracy similar to ARIMA (45.9%).

The Payment dataset confirms this behavior (Figure 8.2d). While the naive forecasts have an SMAPE of 21.8% (N) and 28.5% (NS), reconfirming the observation of the top level, ARIMA (A) is slightly more accurate with 20.8%. The evaluation does not reveal an improvement of CSAR over ARIMA, with an SMAPE ranging between 21.7% (OC and CC) and 21.4% (FC).

Overall, our experiments show that CSAR is at least as accurate as ARIMA on the base level and even more accurate on the top level. Furthermore, the forecasts are calculated much faster, as the next experiment will show.

Model Validation Runtime

To identify ARIMA metaparameters, *auto.arima* needs on average 25 seconds per time series. This sums up to 163,668 seconds, i.e., approximately 45 hours, for all Metering time series (Table 8.3). For CSAR, we assess all metaparameter combinations based on their one-step ahead forecast in the validation interval. For a single time instance and all time series, the validation of OC takes on average 69 seconds with a standard deviation of 4 seconds. Splitting a dataset into clusters leads to a higher validation time of 108 seconds (CC) and 846 seconds (FC) with a similar standard deviation. CC splits the dataset into three clusters but the validation time does not triple compared to OC. Therefore, smaller partitions need less validation time but clustering itself incurs some overhead compared to OC.

This behavior is confirmed on the Payment dataset. ARIMA (A) only needs 1.9 seconds per time series, thus, 3,792 seconds, i.e., approximately 1.1 hours, for all time series and

Table 8.3: Runtime per Time Instance in Seconds

Technique	Validation		Use	
	Metering	Payment	Metering	Payment
A	163,668	3,792	141,968	582
OC	69 ± 4	47 ± 4	0.29 ± 0.03	0.10 ± 0.03
CC	108 ± 3	168 ± 3	0.62 ± 0.03	0.59 ± 0.09
FC	846 ± 7	$1{,}186 \pm 8$	3.63 ± 0.14	3.40 ± 0.27

one time instance. It is faster than the validation time on the Metering dataset due to the shorter time series length. Monolithic CSAR, class-, and feature-based partitioning (OC, CC, FC) only need 47 to 1,186 seconds because they validate all time series in a cluster at once. Overall identifying metaparameters for CSAR is one order of magnitude faster than ARIMA, even on long validation intervals.

Model Use Runtime

During model use, a model is estimated based on the best configuration per time series (A) or per cluster (OC, CC, FC). Subsequently, this model is used to provide a forecast. Estimating an ARIMA model (A) for each Energy time series requires on average 22 seconds and adds up to 141,968 seconds, i.e., approximately 39 hours (Table 8.3). Estimating a model for CSAR for a single time instance only requires 0.29 seconds (OC). Class- and feature-aware clustering incurs some overhead since one CSAR model per cluster is estimated. However, even with clustering, CSAR outperforms ARIMA model estimation by two orders of magnitude.

This behavior is confirmed by the Payment dataset. Modeling an ARIMA model (A) takes 0.29 seconds which adds up to 582 seconds for one dataset. This is faster than the calculation time on the Energy dataset due to the shorter time series length. Again, monolithic CSAR, class-, and feature-based partititioning (OC, CC, FC) are faster by two orders of magnitude. They are less affected by the time series length, but by the amount of clusters to model.

8.4 SUMMARY

In this chapter, we have proposed a feature-based clustering technique for identifying time series clusters with similar correlation features, and evaluated the clustering for time series forecasting. These features were selected based on the observation that CSAR is closely related to ARIMA models, where these features are also applied. The CSAR forecast technique together with clustering showed its capability to forecast big time series datasets more accurately and by two orders of magnitude faster than traditional techniques. These findings were experimentally confirmed on two real-world datasets.

Hartmann et al. also showed the superiority of a monolithic CSAR model over other traditional and vectorized forecast techniques, i.e., triple exponential smoothing, vector autoregression, Croston's method, and hierarchical forecasting [Har18]. As we are outperforming the monolithic CSAR model on the top level, we surely outperform other traditional forecast techniques, too. But other vectorized forecast technique might improve their results by clustering the time series, which is why we recommend investigating this in future work.

9 CONCLUSIONS

In this work, we have found a novel feature-based engineering technique that fulfills the three goals which we have set in the Introduction (Chapter 1). First, it captures important characteristics which provide beneficial results in subsequent data-mining tasks. Second, it successfully tackles the challenges of big time series datasets (Chapter 2). Long time series with a fine granularity are efficiently transformed into a short representation, and the feature-based distance measure efficiently compares the representations. Finally, it is versatile in that it takes the specific requirements from data-mining tasks into account.

In more detail, we have started by surveying engineering techniques based on raw data, shapes, models, and features (Chapter 3). Comparing these techniques by their time and space requirements revealed that features are most promising. The survey also revealed that feature-based engineering techniques are mainly designed for specific data-mining tasks. However, a versatile engineering technique applicable to several tasks is most desirable. Therefore, we studied concrete big time series datasets in a design rationale, and based on these observations, we introduced our feature-based engineering technique (Chapter 4). Its representation efficiently captured deterministic characteristics, i.e., trend and season, as well as stochastic characteristics, i.e., distribution and correlation, using our multi-seasonal decomposition technique. Its distance measure based on the Euclidean distance compared the resulting feature vectors and thus, the structure of the underlying time series. This versatile technique provided the backbone for subsequent techniques.

In the second part of this work, we have successfully applied our feature-based engineering technique to time series generation, matching, classification, and clustering. Regarding time series generation (Chapter 5), our generation technique FBG captured important time series characteristics and evolved time series that were highly similar to given time series. Thus, FBG outperformed state of the art. Regarding time series matching (Chapter 6), we extended the symbolic aggregate approximation (SAX) with features from our representation. The resulting techniques, the season-aware and trend-aware symbolic approximations (sSAX and tSAX), provided a more effective retrieval of similar time series while keeping the same representation size as SAX. Most importantly, we proved that the corresponding distance measures were lower-bounding. Although sSAX increased the runtime for calculating the representation distance, exact matching was up to three orders of magnitude faster due to better pruning. Regarding time series classification (Chapter 7), our feature-based engineering technique tackled the challenges of classifying big datasets with long time series as it efficiently captured discriminative characteristics. Thus, classifiers using our representation achieved both, an excellent classification accuracy as well as a short runtime, compared to state-of-the-art techniques for long time series. Regarding time series clustering (Chapter 8), we devised a meaningful partitioning of time series datasets by using correlation features. Applied to vectorized forecasting, it further improved the forecast accuracy of the cross-sectional autoregression model (CSAR) while keeping a short runtime. In summary, we were able to give better analytical insights by using our versatile feature-based engineering technique.

Future Research Directions

The backbone of this work was an engineering technique based on statistical features. Regarding this foundation and the subsequent data-mining tasks, we see a lot of open challenges and list a few of them that we consider most interesting.

Further Time Series Components Our feature-based representation focuses on a time series model with a linear trend and multiple seasonal components. Other models can be assumed, or other components can be added to the proposed time series model if it is required. Also, non-linear trends may occur. Then, the feature-based representation and distance can be adapted to these requirements. Our symbolic approximations, sSAX and tSAX, focus on one deterministic component per time series, a seasonal component and a linear trend component, respectively. Since time series exhibit these components simultaneously, future studies are required to combine and extend sSAX and tSAX for investigating these cases accurately.

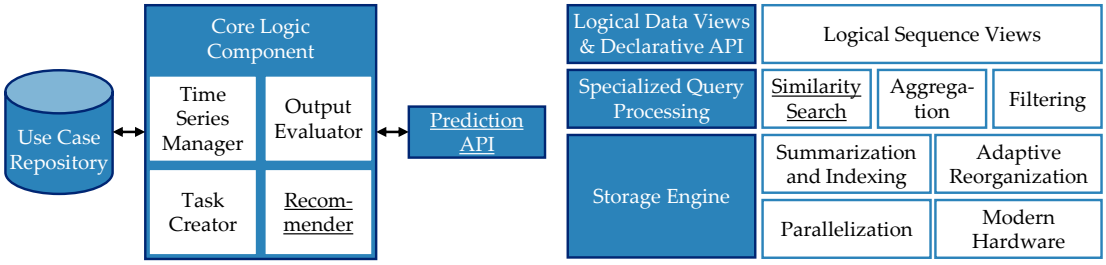
Learned Representations We focused on statistical features that have been presented in the literature on time series analysis. If we consider the automation in many domains, important characteristics should also be extracted automatically. Therefore, further research should focus on learning characteristics with, e.g., ANNs, and compare them with our feature-based representation. As a model-based representation, ANNs require more passes over a dataset for identifying and estimating a model. However, using a model on a time series is fast. Specifically, recurrent networks and auto-encoders [LBH15] have the potential to learn representations for time series classification and clustering. Although they do not provide interpretable features, they should also be applicable to time series generation. As time series matching requires a lower-bounding distance measure, we are not sure whether future work can prove this property for learned representations.

Indexable and Asymmetric Time Series Matching Regarding time series matching, future work should concentrate on time series matching with indexes. Until recently, related work has focused on indexes based on SAX for matching billions of time series. However, the time series lengths are rather short ($T \leq 640$ in [ZIP16], $T \leq 256$ in [ZAER19]). Our season- and trend-aware approximations have the potential to efficiently index and match longer time series thanks to their higher representation accuracy. Moreover, an asymmetric distance measure of sPAA/tPAA with sSAX/tSAX representations should further increase the pruning power and the approximate accuracy [SK08].

Contribution to Analytical Systems

This work has highlighted that feature-based analytics is a versatile tool. In an attempt to give an outlook on its further use, we provide a glance at two analytical systems, *ECAST* and *NESTOR*. Their components are illustrated in Figure 9.1.

The *Energy Forecasting Service* (ECAST) is part of an IoT system for energy load control [GOF17]. For a given forecast task from connected devices, ECAST compares the forecast accuracy of suitable forecast techniques [UFK+14, UHH+16]. Its backend (Figure 9.1a) consists of three main components: a *use case repository*, a *core logic component*, and a *prediction API*. The use case repository stores time series and forecasts, as well as forecast tasks and parameters. The core logic component has a *time series manager* for preparing and transforming the data into the internal format, a *recommender* for retrieving optimal



(a) ECAST [UHH+16] (b) NESTOR [PZ19]

Figure 9.1: Components of Analytical Systems

parameters from the repository, a *task creator* for creating forecast tasks, and an *output evaluator* for assessing the forecast accuracy. Forecast techniques are called via the *prediction API* and return forecasts to the system.

Feature-based analytics has the potential to extend two ECAST components, the recommender and the prediction API. The recommender compares the structural similarity of a given time series and the time series from the repository and suggests parameters from the most similar use case. Feature-based classification could enrich this by suggesting parameters for a specific domain of time series such as the nature of the signal (load, price, wind power production, solar power production) and the aggregation level (household level, grid level). Moreover, vectorized forecast techniques, in combination with feature-based clustering, could extend the prediction API so that datasets of similar time series are forecast together.

Zoumpatianos and Palpanas suggest several courses of action in order to manage big time series datasets and to carry out complex analytics [ZP18]. Their system NESTOR (Figure 9.1b) summarizes these actions in three layers, a *storage engine*, a *specialized query processing*, and a *logical data view* with a *declarative API*. NESTOR's storage engine encompasses techniques for an efficient retrieval of time series, from *indexing* summarized datasets outperforming sequential scans, over an *adaptive reorganization* of the data layout, to efforts regarding *parallelization* and using *modern hardware* such as multi-core, SIMD, and GPU. A specialized query processing enables an efficient time series matching (*similarity search*), as well as an *aggregation* and *filtering* of results. *Logical sequence views* return the results via a declarative interface, taking into account the inherent nature of the data type.

Feature-based analytics could contribute to these actions in two ways. First, our symbolic approximations improve the similarity search; they could be seamlessly integrated as an extension for datasets with deterministic behavior. Second, data management platforms like NESTOR should be assessed using generated datasets with realistic characteristics, as Arlitt et al. suggests [AMB+15]. Our feature-based generation technique evolves characteristics arising in a multitude of domains, and thus, could be useful to achieve this task.

With this in mind, we hope that our research will be valuable in solving the difficulties of current and future analytical systems.

BIBLIOGRAPHY

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient Similarity Search In Sequence Databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, volume 730, pages 69–84, 1993.

[Agg02] Charu C. Aggarwal. On Effective Classification of Strings with Wavelets. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 163–172, 2002.

[AJB97] Henrik André-Jönsson and Dushan Z. Badal. Using Signature Files for Querying Time-Series Data. In *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 211–220, 1997.

[Ald19] Eric Aldrich. *Functions for Computing Wavelet Filters, Wavelet Transforms and Multiresolution Analyses*, 2019. R package version 0.3.0, `https://cran.r-project.org/package=wavelets`.

[AMB+15] Martin F. Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. IoTAbench: an Internet of Things Analytics Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 133–144, 2015.

[APHRH13] Florencia Almonacid, Pedro Pérez-Higueras, Pedro Rodrigo, and Leocadio Hontoria. Generation of ambient temperature hourly time series for some Spanish locations by artificial neural networks. *Renewable Energy*, 51:285–291, 2013.

[ASY15] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering – A decade review. *Information Systems*, 53:16–38, 2015.

[BDHL12] Anthony Bagnall, Luke Davis, Jon Hills, and Jason Lines. Transformation Based Ensembles for Time Series Classification. In *Proceedings of the 12th SIAM International Conference on Data Mining*, pages 307–318, 2012.

[BdMK02] Julia Bilbao, Argimiro H. de Miguel, and Harry D. Kambezidis. Air temperature model evaluation in the north mediterranean belt area. *Journal of Applied Meteorology and Climatology*, 41(8):872 – 884, 2002.

[Bel77] Antonio Bellacicco. Clustering Time Varying Data. In *Proceedings of the European Meeting of Statisticians*, pages 739–747, 1977.

[Bha43] Anil Kumar Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99 – 109, 1943.

[BJ14] Anthony J. Bagnall and Gareth Janacek. A Run Length Transformation for Discriminating Between Auto Regressive Time Series. *Journal of Classification*, 31:154–178, 2014.

[BJR08] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time series analysis: forecasting and control*. Wiley, 4th edition, 2008.

[BK09] Kevin Brokish and James Kirtley. Pitfalls of modeling wind power using markov chains. In *Proceedings of the IEEE/PES Power Systems Conference and Exposition*, pages 1 – 6, March 2009.

[BK15] Matthew Butler and Dimitar Kazakov. SAX Discretization Does Not Guarantee Equiprobable Symbols. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1162–1166, 2015.

[BK17] Martin Becker and Stefan Klößner. *Pearson Distribution System*, 2017. R package version 1.1, `https://cran.r-project.org/package=PearsonDS`.

[BLB+17] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[BLHB15] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[Bur17] U.S. Census Bureau. X-13ARIMA-SEATS Reference Manual V1.1, 2017. `http://census.gov/ts/x13as/docX13ASHTML.pdf` (visited on January 2, 2017).

[BYS08] Depei Bao, Zehong Yang, and Yixu Song. A generalized model for financial time series representation and prediction. *Applied Intelligence*, 29(1):1–11, 2008.

[CBNKL18] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests. *Neurocomputing*, 307:72 – 77, 2018.

[CCL+07] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable PLA for Efficient Similarity Search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 435–446, 2007.

[CCMT90] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[CDB94] Mark A. Cuddihy, J. B. Drummond Jr., and Daniel J. Bourquin. Vehicle Crash Data Generator, 1994.

[CF99] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient Time Series Matching by Wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, pages 126–133, 1999.

[CHB+19] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, and Yutian Li Yifeng Geng. *Extreme Gradient Boosting*, 2019. R package version 0.90.0.2, `https://cran.r-project.org/package=xgboost`.

[CKH+15] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, and Anthony Bagnall. UCR Time Series Classification Archive, 2015.

[CKLF16] Maximilian Christ, Andreas W. Kempa-Liehr, and Michael Feindt. Distributed and parallel time series feature extraction for industrial big data applications. Technical report, Universität Freiburg, 2016.

[CP08] Marcella Corduas and Domenico Piccolo. Time series clustering and classification by the autoregressive metric. *Computational Statistics & Data Analysis*, 52(4):1860–1872, 2008.

[CPSK10] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0: Indexing and Mining One Billion Time Series. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 58–67, 2010.

[Dag80] Estella Bee Dagum. *The X-11-ARIMA seasonal adjustment method*. Statistics Canada, 1980.

[DBB+18] Michael Durstewitz, Guillaume Behem, Volker Berkhout, Elisabeth Buchmann, Robert Cernusko, Stefan Faulstich, Berthold Hahn, Marc-Alexander Lutz, Sebastian Pfaffel, Florian Rehwald, and Susann Spriestersbach. Windenergie Report Deutschland 2017. Fraunhofer Verlag, 2018.

[DOR04] Philip De Chazal, Maria O'Dwyer, and Richard B. Reilly. Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features. *IEEE Transactions on Biomedical Engineering*, 51(7):1196–1206, 2004.

[DPW96] Frank Dellaert, Thomas Polzin, and Alex Waibel. Recognizing emotion in speech. In *Proceedings of the 4th International Conference on Spoken Language Processing*, pages 1970–1973, 1996.

[DTS+08] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[Eur09] The Directive 2009/28/EC of the European Parliament and the Council of the European Union, 2009. `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32009L0028`.

[Eur19] The Renewable Energy Progress Report of the European Commission, 2019. `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52019DC0225`.

[Fal96] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer, 1996.

[FFQ17] The FFQ Project, 2017. `https://wwwdb.inf.tu-dresden.de`.

[FHW16] Eibe Frank, Mark A. Hall, and Ian H. Witten. The WEKA Workbench, 2016. `https://www.cs.waikato.ac.nz/ml/weka`.

[Fis14] Ulrike Fischer. *Forecasting in Database Systems*. PhD thesis, Technische Universität Dresden, 2014.

[FJ14] Ben D. Fulcher and Nick S. Jones. Highly Comparative Feature-Based Time-Series Classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[FLJ13] Ben D. Fulcher, Max A. Little, and Nick S Jones. Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of the Royal Society Interface*, 10(83), 2013.

[FPD+17] Germain Forestier, François Petitjean, Hoang Anh Dau, Geoffrey I. Webb, and Eamonn J. Keogh. Generating synthetic time series to augment sparse datasets. In *Proceedings of the IEEE International Conference on Data Mining*, 2017.

[Fu11] Tak-Chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GHLR01] Cyril Goutte, Lars Kai Hansen, Matthew G. Liptrot, and Egill Rostrup. Feature-Space Clustering for fMRI Meta-Analysis. *Human Brain Mapping*, 13:165–183, 2001.

[Gio19] Toni Giorgino. *Dynamic Time Warping Algorithms*, 2019. R package version 1.21.3, `https://cran.r-project.org/package=dtw`.

[GOF17] The GOFLEX Project, 2017. `https://www.goflex-project.eu`.

[HA13] Robert J. Hyndman and George Athansopoulos. *Forecasting: principles and practice*. OTexts, 2013.

[Har18] Claudio Hartmann. *Forecasting Large-scale Time Series Data*. PhD thesis, Technische Universität Dresden, 2018.

[HK08] Rob J. Hyndman and Yeasmin Khandakar. Automatic Time Series Forecasting: the Forecast Package for R. *Journal Of Statistical Software*, 27(3):1–22, 2008.

[HKL20] Claudio Hartmann, Lars Kegel, and Wolfgang Lehner. Feature-aware forecasting of large-scale time series data sets. *it - Information Technology*, 2020. Published online ahead of print.

[HRH+19] Claudio Hartmann, Franziska Ressel, Martin Hahmann, Dirk Habich, and Wolfgang Lehner. CSAR : the cross-sectional autoregression model for short and long-range forecasting. *International Journal of Data Science and Analytics*, 2019.

[HWN15] Rob J. Hyndman, Earo Wang, and Laptev Nikolay. Large-Scale Unusual Time Series Detection. In *Workshop Proceedings of the IEEE International Conference on Data Mining*, pages 1616–1619, 2015.

[HY99] Yun-Wu Huang and Philip S. Yu. Adaptive Query Processing for Time-Series Data. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 282–286, 1999.

[IFW+19] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[ILD+17] Nadeem Iftikhar, Xiufeng Liu, Sergiu Danalachi, Finn E. Nordbjerg, and Jens H. Vollesen. A scalable smart meter data generator using spark. In *OTM Conferences 2017*, pages 21 – 36, 2017.

[Int17] International Joint Conference on Artificial Intelligence. *IJCAI 2017 - Data Mining Contest*, 2017. `https://tianchi.aliyun.com/competition/entrance/231591/information` (visited on February 8, 2017).

[JL86] D. I. Jones and M. H. Lorenz. An application of a markov chain noise model to wind generator simulation. *Mathematics and Computers in Simulation*, 28(5):391 – 402, 1986.

[KB90] Katarina Košmelj and Vladimir Batagelj. Cross-Sectional Approach for Clustering Time Varying Data. *Journal of Classification*, 7(1):99–109, 1990.

[KDZP18] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. *Proceedings of the VLDB Endowment*, 11(6):677–690, 2018.

[Keg19a] Lars Kegel. Supporting material, 2019. `https://github.com/lkegel/dsax`.

[Keg19b] Lars Kegel. Supporting material, 2019. `https://github.com/lkegel/fbc`.

[Keo97] Eamonn J. Keogh. Fast Similarity Search in the Presence of Longitudinal Scaling in Time Series Databases. In *Proceedings of the 9th International Conference on Tools with Artificial Intelligence*, pages 578–584, 1997.

[KGP01] Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. Distance measures for effective clustering of ARIMA time-series. In *Proceedings of the IEEE International Conference on Data Mining*, 2001.

[KHL16] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Template-based Time series generation with Loom. In *Proceedings of the Workshops of the EDBT/ICDT Joint Conference*, 2016.

[KHL17a] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Feature-driven time series generation. In *Proceedings of the 29th GI-Workshop Grundlagen von Datenbanken*, 2017.

[KHL17b] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Generating What-If Scenarios for Time Series Data. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, 2017.

[KHL18] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Feature-based Comparison and Generation of Time Series. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

[KHSM17] Y. Kang, R. J. Hyndman, and K. Smith-Miles. Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, 33(2):345 – 358, 2017.

[KK03] Eamonn J. Keogh and Shruti Kasetty. On the Need for Time Series Data Mining Benchmarks. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[KKD91] K. M. Knight, S. A. Klein, and J. A. Duffie. A methodology for the synthesis of hourly weather data. *Solar Energy*, 46(2):109 – 120, 1991.

[KKSM91] F. C. Kaminsky, R. H. Kirchhoff, C. Y. Syu, and J. F. Manwell. A comparison of alternative approaches for the synthetic generation of a wind speed time series. *Journal of Solar Energy Engineering*, 113(4):280 – 289, 1991.

[Kol08] Michael Kolonko. Inversionsmethode. In *Stochastische Simulation*, pages 85–95. Vieweg+Teubner, 2008.

[KPMP01] Eamonn Keogh, Michael Pazzani, Sharad Mehrotra, and Michael Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 151–162, 2001.

[KS83] M. Kendall and A. Stuart. *The Advanced Theory of Statistics*, volume 3, pages 410 – 414. Griffin, 1983.

[LBCSA11] Jason Lines, Anthony Bagnall, Patrick Caiger-Smith, and Simon Anderson. Classification of Household Devices by Electricity Usage Profiles. In *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning*, pages 403–412, 2011.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[Lex19] Lexico.com. Oxford Dictionary: Engineering, 2019. Retrieved June 6, 2019, from `https://www.lexico.com/en/definition/engineering`.

[LFK+14] Heiner Lasi, Peter Fettke, Hans Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business and Information Systems Engineering*, 6(4):239–242, 2014.

[Lia05] Thunshun Warren Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38:1857–1874, 2005.

[LKL03] Sangjun Lee, Dongseop Kwon, and Sukho Lee. Dimensionality Reduction for Indexing Time Series Based on the Minimum Distance. *Journal of Information Science and Engineering*, 19(4):697–711, 2003.

[LKLC03] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.

[LKWL07] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[LL18] Yun Li and Tao Li. Feature Engineering for Machine Learning and Data Analytics. In Guozhu Dong and Huan Liu, editors, *Feature Engineering for Machine Learning and Data Analytics*, pages 191–220. CRC Press, 2018.

[LSK06] Battuguldur Lkhagva, Yu Suzuki, and Kyoji Kawagoe. New Time Series Data Representation ESAX for Financial Applications. In *Proc. of the 22nd International Conference on Data Engineering Workshops*, 2006.

[Mah00] Elizabeth Ann Maharaj. Cluster of Time Series, 2000.

[McC02] Peter McCullagh. What is a statistical model? *Annals of Statistics*, 30(5):1225–1267, 2002.

[MDC19] Elizabeth Ann Maharaj, Pierpaolo D'Urso, and Jorge Caiado. *Time Series Clustering and Classification*. Taylor and Francis, 2019.

[MDH+19] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch, Chih-Chung Chang, and Chih-Chen Lin. *Misc Functions of the Department of Statistics, Probability Theory Group*, 2019. R package version 1.7.1, `https://cran.r-project.org/package=e1071`.

[MGQT13] Simon Malinowski, Thomas Guyet, René Quiniou, and Romain Tavenard. 1d-SAX: A Novel Symbolic Representation for Time Series. In *Proceedings of the 12th International Symposium on Intelligent Data Analysis*, pages 273–284, 2013.

[MH00] Spyros Makridakis and Michèle Hibon. The M3-Competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451 – 476, 2000.

[MH15] Hannes Müller and Uwe Haberlandt. Temporal rainfall disaggregation with a cascade model: From single-station disaggregation to spatial rainfall. *Journal of Hydrologic Engineering*, 20(11), 2015.

[MLW04] Vasileios Megalooikonomou, Guo Li, and Qiang Wang. A Dimensionality Reduction Technique for Efficient Similarity Analysis of Time Series Databases. In *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management*, pages 160–161, 2004.

[Mör03] Fabian Mörchen. Time series feature extraction for data mining using DWT and DFT. Technical report no. 33, Philips-Universität Marburg, 2003.

[MS82] B. McWilliams and Dan Sprevak. The simulation of hourly wind speed and direction. *Mathematics and Computers in Simulation*, 24(1):54–59, 1982.

[MSA18] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The M4 Competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802 – 808, 2018.

[MTV+17] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. TimeNet: Pre-trained deep recurrent neural network for time series classification. *Proceedings of the 25th European Symposium on Artificial Neural Networks*, 2017.

[MW01] Polly Wan Po Man and Man Hon Wong. Efficient and Robust Feature Extraction and Pattern Matching of Time Series by a Lattice Structure. In *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management*, pages 271–278, 2001.

[NAM01] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Feature-based Classification of Time-series Data. In *Information processing and technology*, pages 49–61. Nova Science Publishers, 2001.

[PCH93] Steven M. Pincus, Theodore R. Cummins, and Gabriel G. Haddad. Heart rate control in normal and aborted-SIDS infants. *American Journal of Physiology*, 264(3 Pt 2):R638–R646, 1993.

[PF02] Kevin B. Pratt and Eugene Fink. Search for Patterns in Compressed Time Series. *International Journal of Image and Graphics*, 2(1):89–106, 2002.

[PG15] John Paparrizos and Luis Gravano. k-Shape: Efficient and Accurate Clustering of Time Series. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1855–1870, 2015.

[Pic90] D. Piccolo. A distance measure for classifying ARMA models. *Journal of time series analysis*, 11(2):153–163, 1990.

[PO94] Duc Truong Pham and Ercan Oztemel. Control chart pattern recognition using feature-based learning vector quantization. *International Journal of Production Research*, 32(3):721–729, 1994.

[PSAH15] T. Pesch, S. Schröders, H. J. Allelein, and J. F. Hake. A new Markov-chain-related statistical approach for modelling synthetic wind power time series. *New Journal of Physics*, 17(5), 2015.

[PWZP00] Chang-Shing Perng, Haixun Wang, Sylvia R. Zhang, and D. Stott Parker. Landmarks: A New Model for Similarity-Based Pattern Querying in Time Series Databases. In *Proceedings of the 16th International Conference on Data Engineering*, pages 33–42, 2000.

[PZ19] Themis Palpanas and Kostas Zoumpatianos. Storage and retrieval system for complex analytics on big sequence collections, 2019. `http://nestordb.com` (visited on December 12, 2019).

[QWW98] Yunyao Qu, Changzhou Wang, and X. Sean Wang. Supporting Fast Search in Time Series for Movement Patterns in Multiple Scales. In *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management*, pages 251–258, 1998.

[R C18] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018.

[RCM+12] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012.

[RSC02] Marco Ramoni, Paola Sebastiani, and Paul Cohen. Bayesian Clustering by Dynamics. *Machine Learning*, 47(1):91–121, 2002.

[SBW+05] A. Shamshad, M. A. Bawadi, W. M. A. Wan Hussin, T. A. Majid, and S. A. M. Sanusi. First and second order Markov chain models for synthetic generation of wind speed time series. *Energy*, 30(5):693–708, 2005.

[SC78] Hiroaki Sakoe and Seibi Chiba. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 26(1):43, 1978.

[SC07] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[SJ13] Jan Schaffner and Tim Januschowski. Realistic tenant traces for enterprise DBaaS. In *Workshops Proceedings of the 29th IEEE International Conference on Data Engineering*, pages 29 – 35, 2013.

[SK08] Jin Shieh and Eamonn J. Keogh. iSAX: Indexing and Mining Terabyte Sized Time Series. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[SLL+14] Youqiang Sun, Jiuyong Li, Jixue Liu, Bingyu Sun, and Christopher Chow. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing*, 138:189–198, 2014.

[SS11] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2011.

[Åst69] K. J. Åström. On the Choice of Sampling Rates in Parametric Identification of Time Series. *Information Sciences*, 1(3):273–278, 1969.

[STK+03] Michael Steinbach, Pang-Ning Tan, Vipin Kumar, Steven Klooster, and Christopher Potter. Discovery of Climate Indices using Clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 446–455, 2003.

[TAR19] Terry Therneau, Beth Atkinson, and Brian Ripley. *Recursive Partitioning and Regression Trees*, 2019. R package version 4.1.15, `https://cran.r-project.org/package=rpart`.

[TGDH93] J. Timmer, C. Gantert, G. Deuschl, and J. Honerkamp. Characteristics of hand tremor time series. *Biological Cybernetics*, 70:75–80, 1993.

[The11] Marina Theodosiou. Forecasting monthly and quarterly time series using STL decomposition. *International Journal of Forecasting*, 27(4):1178 – 1195, 2011.

[The15] The Commission for Energy Regulation. CER Smart Metering Project, 2015. `www.ucd.ie/issda`.

[TW02] Dat Tran and Michael Wagner. Fuzzy C-Means Clustering-Based Speaker Verification. In *Proceedings of the International Conference on Fuzzy Systems and Advances in Soft Computing*, pages 318–324, 2002.

[UFK+14] Robert Ulbricht, Ulrike Fischer, Lars Kegel, Dirk Habich, Hilko Donker, and Wolfgang Lehner. ECAST: A benchmark Framework for Renewable Energy Forecasting Systems. In *Proceedings of the Workshops of the EDBT/ICDT Joint Conference*, 2014.

[UFLD13] Robert Ulbricht, Ulrike Fischer, Wolfgang Lehner, and Hilko Donker. Rethinking Energy Data Management: Trends and Challenges in Today's Transforming Markets. In *Proceedings der 15. BTW Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme*, pages 421–440, 2013.

[UHH+16] Robert Ulbricht, Claudio Hartmann, Martin Hahmann, Hilko Donker, and Wolfgang Lehner. Web-based Benchmarks for Forecasting Systems - The ECAST Platform. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 2169–2172, 2016.

[vdMH18] Laurens J. P. van der Maaten and Geoffrey E. Hinton. Visualizing Data Using t-SNE. *Journal of Machine Learning Research*, 26(9):2579–2605, 2018.

[VGK02] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. Discovering Similar Multidimensional Trajectories. In *Proceedings of the 18th International Conference on Data Engineering*, pages 673–684, 2002.

[vPL02] A. H. C. van Paassen and Q. X. Luo. Weather data generator to study climate change on buildings. *Building Services Engineering Research and Technology*, 23(4):251 – 258, 2002.

[WGH12] Jenna Wiens, John V. Guttag, and Eric Horvitz. Patient Risk Stratification for Hospital-Associated C. diff as a Time-Series Classification Task. In *Advances in Neural Information Processing Systems 25*, volume 1, pages 476 – 484, 2012.

[WSH06] Xiaozhe Wang, Kate Smith, and Rob Hyndman. Characteristic-Based Clustering for Time Series Data. *Data Mining and Knowledge Discovery*, 13(3):335–364, 2006.

[XPK10] Zhengzheng Xing, Jian Pei, and Eamonn J. Keogh. A Brief Survey on Sequence Classification. *SIGKDD Explorations*, 12(1):40, 2010.

[XY02] Yimin Xiong and Dit-Yan Yeung. Proceedings of the IEEE International Conference on Data Mining. In *ICDM*, pages 717–720, 2002.

[XYWV12] Feng Xia, Laurence T. Yang, Lizhe Wang, and Alexey Vinel. Internet of Things. *International Journal of Communication Systems*, 25(9):1101–1102, 2012.

[YF00] Byoung-Kee Yi and Christos Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[YK09] Lexiang Ye and Eamonn Keogh. Time Series Shapelets: A New Primitive for Data Mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 947–955, 2009.

[YYZ+15] Hong Yin, Shuqiang Yang, Xiaoqian Zhu, Shaodong Ma, and Lumin Zhang. Symbolic representation based on trend features for biomedical data classification. *Frontiers of Information Technology & Electronic Engineering*, 16(9):744–758, 2015.

[ZAER19] Liang Zhang, Noura Alghamdi, Mohamed Y. Eltabakh, and Elke A. Rundensteiner. TARDIS : Distributed Indexing Framework for Big Time Series Data. In *Proceedings of the 35th IEEE International Conference on Data Engineering*, 2019.

[ZIP16] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. ADS: the adaptive data series index. *VLDB Journal*, 25(6):843–866, 2016.

[ZP18] Kostas Zoumpatianos and Themis Palpanas. Data Series Management : Fulfilling the Need for Big Sequence Analytics. In *Proceedings of the 34th IEEE International Conference on Data Engineering*, pages 16–17, 2018.

[ZY16] Chaw Thet Zan and Hayato Yamana. An improved symbolic aggregate approximation distance measure based on its statistical features. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pages 72–80, 2016.

LIST OF FIGURES

1.1 Time Series Analytics . . . 12
1.2 Structure of this Work . . . 14

2.1 Domains of Time Series . . . 16
2.2 Data-mining Tasks . . . 18

3.1 Taxonomy of Engineering Techniques . . . 24
3.2 Time Series With PAA and SAX Representations . . . 28
3.3 Life cycle of Time Series Model . . . 30

4.1 Multi-seasonal Decomposition . . . 46
4.2 Non-normalized Features (Metering) . . . 48
4.3 Non-normalized Features (Wind) . . . 49
4.4 Normalized Features . . . 50

5.1 Properties of Generation Techniques . . . 53
5.2 Overview of Feature-based Generation . . . 58
5.3 Modification of Time Series Components . . . 60
5.4 Feature Space of Economy Dataset . . . 62
5.5 Autoregressive Model . . . 63
5.6 Feature-based Distance of Recombination . . . 67
5.7 Feature-based Distance of Markov Chain . . . 67
5.8 Feature-based Distance of Genetic Algorithm . . . 67
5.9 Visual Distances . . . 70

6.1 Overview of Time Series Matching . . . 74
6.2 Time Series with Season . . . 79
6.3 Advantages of Season-aware SAX over SAX . . . 80
6.4 Time Series with Trend . . . 82
6.5 Entropy on Season . . . 88
6.6 Entropy on Trend . . . 88
6.7 Increase of TLB compared to SAX . . . 89
6.8 Increase of Pruning Power Compared to SAX . . . 89
6.9 Increase of Approximate Accuracy Compared to SAX . . . 89

7.1 Run Length Distribution . . . 95
7.2 Features from tsfresh . . . 96
7.3 Overview of Feature-based Classification . . . 97
7.4 Feature-based Representation of Metering Dataset as Heatmap . . . 98
7.5 Correlation-based Feature Selection [LL18] . . . 99
7.6 Configuration of FBR . . . 104
7.7 Configuration of tsfresh . . . 105
7.8 Configuration of DWT . . . 106
7.9 Classification Accuracy . . . 106
7.10 Overall Runtime . . . 107
7.11 Classification Accuracy versus Overall Runtime . . . 108

8.1 Clustering of Metering Dataset 115
8.2 Forecast Accuracy 117

9.1 Components of Analytical Systems 121

LIST OF TABLES

3.1 Properties of Engineering Techniques 40

4.1 Properties of Decomposition Techniques 44
4.2 Mean Runtime per Time Series in Seconds (Metering) 48

5.1 Properties of Generation Techniques 57
5.2 Properties of Proposed Generation Techniques 64
5.3 Experimental Setting 66
5.4 Median Feature-based Distance on All Datasets 69
5.5 Standard Distances 71

6.1 Properties of Engineering Techniques 77
6.2 Dataset Dimensions 85
6.3 Configurations of Engineering Techniques 87
6.4 Matching Efficiency on Season (Large) 90

7.1 Model Training Runtime of COTE 94
7.2 Haar Wavelet 95
7.3 Datasets for Classification 103

8.1 Correlation Features for CSAR 113
8.2 Metaparameter and Time Intervals 115
8.3 Runtime per Time Instance in Seconds 118

A

PROOFS FOR SSAX AND TSAX

A.1 PROOF OF LOWER-BOUNDING SPAA

Proof. We show that the following inequality always holds:

$$d_{sPAA}(\bar{\underline{y}}_{sPAA}, \bar{\underline{y}}'_{sPAA}) \leq d_{ED}(\underline{y}, \underline{y}') \tag{A.1}$$

For convenience, we use $E = {}^{T}/_{W}$ for the length of a segment and $B = {}^{T}/_{L}$ for the number of seasons in the time series. We square both sides:

$$d_{sPAA}(\bar{\underline{y}}_{sPAA}, \bar{\underline{y}}'_{sPAA})^2 \leq d_{ED}(\underline{y}, \underline{y}')^2 \tag{A.2}$$

Until Equation A.11, we focus on transforming the left-hand side. Using d_{sPAA} from Equation 6.15 (page 80) yields:

$$\frac{T}{W \cdot L} \sum_{t=1}^{T} (\Delta\sigma_{(t-1)\%L+1} + \Delta\overline{res}_{\lfloor \frac{t-1}{E} \rfloor + 1})^2 \tag{A.3}$$

Rewriting the residuals with Equation 3.7 (page 27) from PAA yields:

$$\frac{T}{W \cdot L} \sum_{t=1}^{T} (\Delta\sigma_{(t-1)\%L+1} + \frac{1}{E} \sum_{j=1}^{E} \Delta res_{\lfloor \frac{t-1}{E} \rfloor \cdot E + j})^2 \tag{A.4}$$

The residuals are the difference of time series values and season mask:

$$\begin{aligned} \frac{T}{W \cdot L} \sum_{t=1}^{T} (& + \Delta\sigma_{(t-1)\%L+1} \\ & + \frac{1}{E} \sum_{j=1}^{E} \Delta y_{\lfloor \frac{t-1}{E} \rfloor \cdot E + j} \\ & - \frac{1}{E} \sum_{j=1}^{E} \Delta\sigma_{(\lfloor \frac{t-1}{E} \rfloor \cdot E + j - 1)\%L+1})^2 \end{aligned} \tag{A.5}$$

The season mask is also expressed as mean values of $\underline{y}$ (Equation 6.10, page 78):

$$\frac{T}{W \cdot L} \sum_{t=1}^{T} \Big(+ \frac{1}{B} \sum_{k=1}^{B} \Delta y_{(k-1)\cdot L+(t-1)\%L+1} + \frac{1}{E} \sum_{j=1}^{E} \Delta y_{\lfloor \frac{t-1}{E} \rfloor \cdot E+j} - \frac{1}{B} \frac{1}{E} \sum_{j=1}^{E} \sum_{k=1}^{B} \Delta y_{(k-1)\cdot L+(\lfloor \frac{t-1}{E} \rfloor \cdot E+j-1)\%L+1} \Big)^2 \tag{A.6}$$

We focus on the case when $W \cdot L = T$, thus $E = L$ and $W = B$. Factoring out $\frac{1}{B \cdot L} = \frac{1}{T}$ yields:

$$\frac{1}{T^2} \sum_{t=1}^{T} \Big(+ L \cdot \sum_{k=1}^{B} \Delta y_{(k-1)\cdot L+(t-1)\%L+1} + B \cdot \sum_{j=1}^{L} \Delta y_{\lfloor \frac{t-1}{L} \rfloor \cdot L+j} - \sum_{j=1}^{L} \sum_{k=1}^{B} \Delta y_{(k-1)\cdot L+j} \Big)^2 \tag{A.7}$$

We arrange the distances Δy_t in a matrix $D \in \mathbb{R}^{B \times L}$ as follows:

$$D = \begin{pmatrix} \Delta y_1 & \Delta y_2 & \dots & \Delta y_L \\ \Delta y_{L+1} & \Delta y_{L+2} & \dots & \Delta y_{2\cdot L} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta y_{(B-1)\cdot L+1} & \Delta y_{(B-1)\cdot L+2} & \dots & \Delta y_T \end{pmatrix} \tag{A.8}$$

Consequently, we arrange the factors L and B and -1 in a matrix $A^t \in \mathbb{R}^{B \times L}$. The positions of these factors depend on t. The cell of t which is in row $\lfloor \frac{t-1}{L} \rfloor + 1$ and column $(t-1)\%L+1$ is filled with $B+L-1$. The other cells in the same row are filled with $B-1$. The other cells in the same column are filled with $L-1$. All other cells are filled with -1. For example, for $t = 1$:

$$A^1 = \begin{pmatrix} B+L-1 & B-1 & \dots & B-1 \\ L-1 & -1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ L-1 & -1 & \dots & -1 \end{pmatrix} \tag{A.9}$$

With these two matrices, Equation A.7 is represented as follows:

$$\frac{1}{T^2} \sum_{t=1}^{T} \Big(\sum_{k=1}^{B} \sum_{j=1}^{L} A^t_{j,k} D_{j,k} \Big)^2 \tag{A.10}$$

Expanding the square and iterating through t yields:

$$\begin{aligned}\frac{1}{T}(&+(B+L-1)\sum_{\substack{1\le k\le B\\1\le j\le L}} D_{k,j}^2\\&+(B-1)\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le j'\le L\\j'\neq j}} D_{k,j}\cdot D_{k,j'}\\&+(L-1)\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le k'\le B\\k'\neq k}} D_{k,j}\cdot D_{k',j}\\&-\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le k'\le B\\1\le j'\le L\\k'\neq k\\j'\neq j}} D_{k,j}\cdot D_{k',j'})\end{aligned} \tag{A.11}$$

The Euclidean distance from the right-hand side (Equation A.2) can be rewritten as $d_{ED}(\underline{y},\underline{y}')^2 = \sum_{\substack{1\le k\le B\\1\le j\le L}} D_{k,j}^2$ and can be subtracted from the left-hand side. Multiplying by $-T$ leads to:

$$+(B\cdot L+1-B-L)\sum_{\substack{1\le k\le B\\1\le j\le L}} D_{k,j}^2 \tag{A.12}$$

$$-(1-B)\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le j'\le L\\j'\neq j}} D_{k,j}\cdot D_{k,j'} \tag{A.13}$$

$$-(1-L)\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le k'\le B\\k'\neq k}} D_{k,j}\cdot D_{k',j} \tag{A.14}$$

$$+\sum_{\substack{1\le k\le B\\1\le j\le L}}\sum_{\substack{1\le k'\le B\\1\le j'\le L\\k'\neq k\\j'\neq j}} D_{k,j}\cdot D_{k',j'}\ge 0 \tag{A.15}$$

This is equivalent to:

$$\sum_{k=1}^{B}\sum_{\substack{k'=1\\k<k'}}^{B}\sum_{j=1}^{L}\sum_{\substack{j'=1\\j<j'}}^{L}(D_{k,j}-D_{k,j'}-D_{k',j}+D_{k',j'})^2\ge 0 \tag{A.16}$$

Since all summands are squared, they are always greater than or equal to zero. The other cases $W\cdot L\neq T$ where $W\cdot L$ divides T can be solved similarily. □

A.2 PROOF OF LOWER-BOUNDING SSAX

Proof. We show that sSAX lower-bounds the Euclidean distance:

$$d_{sSAX}(\hat{\underline{y}}_{sSAX},\hat{\underline{y}}'_{sSAX})\le d_{ED}(\underline{y},\underline{y}') \tag{A.17}$$

We show this property based on the distance measure from Equation 6.18 (page 81). For given symbols $\hat{\sigma},\hat{\sigma}',\widehat{res},\widehat{res}'$, we assume that the Case 1 from Equation 6.18 holds:

$c_s(\hat{\sigma}, \hat{\sigma}') \geq -c_s(\widehat{res}, \widehat{res}')$. Using Equation 6.17 (page 80) and reformulating the inequality taking into account the sPAA features yields:

$$\sigma + \overline{res} \geq b_{\hat{\sigma}} + b_{\widehat{res}} \geq b_{\hat{\sigma}'+1} + b_{\widehat{res}'+1} \geq \sigma' + \overline{res}' \tag{A.18}$$

Thus, $cell(\hat{\sigma}, \hat{\sigma}', \widehat{res}, \widehat{res}')$ is always lower than the sum of sPAA features:

$$\begin{aligned} & cell(\hat{\sigma}, \hat{\sigma}', \widehat{res}, \widehat{res}') \\ & \quad = c_s(\hat{\sigma}, \hat{\sigma}') + c_s(\widehat{res}, \widehat{res}') \\ & \quad = b_{\hat{\sigma}} - b_{\hat{\sigma}'+1} + b_{\widehat{res}} - b_{\widehat{res}'+1} \\ & \quad = b_{\hat{\sigma}} + b_{\widehat{res}} - (b_{\hat{\sigma}'+1} + b_{\widehat{res}'+1}) \\ & \quad \leq \sigma + \overline{res} - (\sigma' + \overline{res}') \end{aligned} \tag{A.19}$$

By symmetry, this inequality also holds for Case 2 of Equation 6.18 and it is trivial for Case 3. Thus, we can conclude:

$$d_{sSAX}(\hat{\underline{y}}_{sSAX}, \hat{\underline{y}}'_{sSAX}) \leq d_{sPAA}(\underline{\bar{y}}_{sPAA}, \underline{\bar{y}}'_{sPAA}) \tag{A.20}$$

As shown in Section A.1, sPAA distance lower-bounds the Euclidean distance. □

A.3 PROOF OF COMBINED TREND FEATURE

Proof. We prove Equation 6.23 (page 81) as follows. As defined in Chapter 6, a time series for time series matching is z-normalized. Thus, the mean of its values is zero. Taking into account the trend component (Equations 6.19 and 6.20, page 81):

$$\sum_{t=1}^{T} y_t = \sum_{t=1}^{T} (tr_t + res_t) = \sum_{t=1}^{T} (\theta_1 + \theta_2 \cdot (t-1) + res_t) = 0 \tag{A.21}$$

Linear regression implies that the residuals' sum is zero (Equation 6.21, page 81):

$$\sum_{t=1}^{T} (\theta_1 + \theta_2 \cdot (t-1)) = T \cdot (\theta_1 + \theta_2 \cdot \frac{T-1}{2}) = 0 \tag{A.22}$$

□

A.4 PROOF OF LOWER-BOUNDING TPAA

Proof. We show that tPAA lower-bounds the Euclidean distance:

$$d_{tPAA}(\underline{\bar{y}}_{tPAA}, \underline{\bar{y}}'_{tPAA}) \leq d_{ED}(\underline{y}, \underline{y}') \tag{A.23}$$

For convenience, we rewrite $\overline{res}_t = \overline{res}_{\lfloor (t-1)/(T/W) \rfloor + 1}$. Squaring each side and setting the trend components yields:

$$\sum_{t=1}^{T} (\Delta tr_t + \Delta \overline{res}_t)^2 \leq \sum_{t=1}^{T} (\Delta tr_t + \Delta res_t)^2 \tag{A.24}$$

Expanding the summands and subtracting the common summands:

$$\sum_{t=1}^{T}\left((\Delta\overline{res}_t)^2 + 2\cdot\Delta tr_t\cdot\Delta\overline{res}_t\right) \leq \sum_{t=1}^{T}\left((\Delta res_t)^2 + 2\cdot\Delta tr_t\cdot\Delta res_t\right) \tag{A.25}$$

All summands except one are arranged on the right-hand side:

$$\sum_{t=1}^{T}(\Delta\overline{res}_t)^2 \leq \sum_{t=1}^{T}\left((\Delta res_t)^2 + 2\cdot\Delta tr_t\cdot(\Delta res_t - \Delta\overline{res}_t)\right) \tag{A.26}$$

The trend component and the residuals are not correlated which is also true for mean residuals (Equation 6.22, page 81). Thus, this equation can be rewritten as:

$$\sum_{t=1}^{T}(\Delta\overline{res}_t)^2 \leq \sum_{t=1}^{T}(\Delta res_t)^2 \tag{A.27}$$

PAA distance lower-bounds the Euclidean distance as shown in [YF00]. □

A.5 PROOF OF LOWER-BOUNDING TSAX

Proof. We show that tSAX lower-bounds the Euclidean distance:

$$d_{tSAX}(\hat{\underline{y}}_{tSAX}, \hat{\underline{y}}'_{tSAX}) \leq d_{ED}(\underline{y}, \underline{y}') \tag{A.28}$$

By construction, the lookup table c_t always returns the minimum distance between deterministic components represented by $\hat{\phi}$ and by $\hat{\phi}'$. Similarily, the lookup table $cell$ returns the minimum distance between two PAA mean values $\widehat{\overline{res}}_w$ and $\widehat{\overline{res}}'_w$. Therefore:

$$\begin{aligned}
&d_{tSAX}(\hat{\underline{y}}_{tSAX}, \hat{\underline{y}}'_{tSAX}) && \text{(A.29)}\\
&\quad= \sqrt{c_t(\hat{\phi},\hat{\phi}')^2 + \frac{T}{W}\sum\nolimits_{w=1}^{W} cell(\widehat{\overline{res}}_w, \widehat{\overline{res}}'_w)^2} && \text{(A.30)}\\
&\quad\leq \sqrt{\sum\nolimits_{t=1}^{T}((\Delta tr_t)^2 + (\Delta\overline{res}_t)^2)} && \text{(A.31)}\\
&\quad= \sqrt{\sum\nolimits_{t=1}^{T}\left((\Delta tr_t)^2 + (\Delta\overline{res}_t)^2 + 2\cdot\Delta tr_t\cdot\Delta\overline{res}_t\right)} && \text{(A.32)}\\
&\quad= \sqrt{\sum\nolimits_{t=1}^{T}(\Delta tr_t + \Delta\overline{res}_t)^2} && \text{(A.33)}\\
&\quad= d_{tPAA}(\overline{\underline{y}}_{tPAA}, \overline{\underline{y}}'_{tPAA}) && \text{(A.34)}
\end{aligned}$$

In Equation A.32, we introduce a summand that equals zero because trend and residual components are not correlated (Equation 6.22, page 81). As already shown, tPAA distance lower-bounds the Euclidean distance (Appendix A.4). □

B

LIST OF SYMBOLS

We summarize all symbols that have been used in this work. Since we refer to different methods of the time-series analysis literature, some symbols are non-unique. We changed the original notation if the meaning was ambiguous, otherwise we left it.

Latin Symbols

a (ARIMA model)	non-seasonal error term
a (discretization)	symbol
A	size of alphabet of symbols, states or histogram buckets
AA	approximate accuracy
acc	classification accuracy
acf	autocorrelation
b	breakpoint
B	backshift operator
c	index of class, i.e., class label
C	number of classes
cc	Pearson correlation coefficient
$cell$	lookup table
cov	covariance
c_s	lookup table
d (ARIMA model)	order of non-seasonal differentiation
d (distance measure)	distance function
D	order of seasonal differentiation
det	deterministic value
e	CSAR error term
f (feature-based engineering)	feature
f (statistical model)	probability density function
F (feature-based engineering)	value set of one feature
F (statistical model)	cumulative distribution function
$freq$	relative frequency of values within histogram bucket
g	label function
G	season granularity

h	lag of autocorrelation, partial autocorrelation, or value set
H	entropy
i	index of time series in dataset, a state, or a histogram bucket
I (foundations)	size of time series dataset
I (function)	indicator function
j	index of a second state or a recombination candidate
k	index of a feature or a DTW warping path element
K	length of feature vector, a value set, or a DTW warping path
$kurt$	kurtosis
L (feature-based engineering)	season length
L (function)	likelihood function
m	mean value
$minPts$	DBSCAN clustering parameter
n	absolute frequency
$\mathcal{N}$	normal distribution
o	order of Minkowski distance
p (ARIMA model)	order of non-seasonal autoregressive process or cross section
p (feature-based engineering)	lower threshold
p (function)	probability function
P (ARIMA model)	order of seasonal autoregressive process or cross section
P (Markov model)	transition probability matrix
P (statistical model)	probability distribution
$\mathcal{P}$	set of probability distributions
$pacf$	partial autocorrelation
PP	pruning power
q	order of non-seasonal moving average process or amount of non-seasonal CSAR error terms
q	(feature-based engineering) upper threshold
Q	order of seasonal moving average process or amount of seasonal CSAR error terms
R^2	strength of component
res	residual value
s	index of seasonal component
S	amount of seasonal components
$\mathcal{S}$	sample space
$SAPE$	symmetric absolute percentage error (accuracy measure)
sd	standard deviation
$seas$	season value
$skew$	skewness
t	time instance
T	length of time series
TLB	tightness of lower bound
tr	trend value
Tr	number of time instances for model training

U	number of time instances or time series for model use
$\mathcal{U}$	uniform distribution
v	differentiated time series value
V	number of time instances for model validation
var	variance
w	index of segment
W	amount of scalars in a representation
WP	DTW warping path
y	time series value
$\underline{y_i}$	time series
$\underline{y_t}$	cross section
Y (foundations)	time series dataset
Y (model-based engineering)	random variable
Z	random variable

Greek Symbols

α	seasonal error term
β	factor of generation technique MD
γ	factor of generation technique MD
δ	local trend change value
Δ	difference operator
ϵ	DBSCAN clustering parameter
θ (ARIMA model)	weight of non-seasonal moving average process
θ (statistical model)	set of model parameters
Θ (ARIMA model)	weight of seasonal moving average process
Θ (statistical model)	space of model parameters
θ_1	base value
θ_2	trend slope
γ	factor of generation technique MD
μ	mean (model parameter)
π	weight of linear process regressed on itself
σ	seasonal feature
σ^2	variance (model parameter)
ϕ (ARIMA model)	weight of non-seasonal autoregressive process
ϕ (feature-based engineering)	trend feature
Φ	weight of seasonal autoregressive process
ψ	weight of linear process

Accents and Other Symbols

N. B. x is a placeholder for different elements such as a value (y), a feature (f), etc.

$\underline{x}$	a vector of elements x
$\bar{x}$ (shape-based engineering)	reduction to a real value
$\bar{\underline{x}}$ (time series classification)	subset from $\underline{x}$
$\hat{x}$ (model-based engineering)	estimated model parameter

$\hat{x}$ (shape-based engineering)	discretization
$\hat{x}$ (time series clustering)	forecast
$\tilde{x}$	element corrected by its mean or base value
x^n	normalized element
x^*	target element

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, January 9, 2020