Facilitating and Enhancing the Performance of Model Selection for Energy Time Series Forecasting in Cluster Computing Environments

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc. Shadi Shahoud

Tag der mündlichen Prüfung:	02.12.2022
Erster Gutachter:	Prof. Dr. Veit Hagenmeyer
Zweiter Gutachter:	Prof. Dr. rer. nat. Florian Steinke

Acknowledgement

This dissertation is the result of a very enjoyable challenging journey. Finishing it would not have been possible without the constant support and assistance of many people.

First of all, I would like to give my deepest gratitude to my supervisor Prof. Veit Hagenmeyer. Thank you for your guidance, trust, advice and continuous support. Thank you for granting me the opportunity to carry out my research at the institute. I would also like to extend my sincere thanks to Prof. Florian Steinke for agreeing to be my external reviewer and providing feedback.

Thank you Clemens Düpmeier for your inspiring ideas, valuable suggestions and unbounded patience while discussing and building this research from ground. Working directly with you helped me to deepen my knowledge in the research field through the continuous exchange of ideas. For you, I am incredibly thankful. Thank you Kevin Förderer for your review and numerous suggestions that helped to improve the quality of my dissertation.

I want to thank all my students with whom I had the pleasure to work: Sonja Gunnarsdottir, Moritz Winter and Adrian Beer. Supervising you has been the most rewarding part of my job. I hope I was able to give you as much as you gave to me.

My gratitude also goes to the old friend and best office mate Hatem Khalloof. Thank you for all discussions, collaborative work, laughter, guidance and support you have given me and, yes, also for your cheerful mood, for having listened to my imagination running, and for having advised me on how to bring imagination back to science. Thanks also to my colleagues at IAI for the great past five years, especially Dominique, Eric, Richard, Rafael, Christian, Claudia, Thorsten, Christina, Jianlei and Jannik.

I would like to thank my friends, for always believing in me. Thank you to my family for everything you gave me, without which I could not be here writing this dissertation. Thank you for the love, encouragement and unwavering support over the years. Thank you for having been always present, even when I was far and distracted. I am forever indebted.

Karlsruhe, December 2022 Shadi Shahoud

Abstract

Applying Machine Learning (ML) manually to a given problem setting is a tedious and time-consuming process which brings many challenges with it, especially in the context of Big Data. In such a context, gaining insightful information, finding patterns, and extracting knowledge from large datasets are quite complex tasks. Additionally, the configurations of the underlying Big Data infrastructure introduce more complexity for configuring and running ML tasks. With the growing interest in ML the last few years, particularly people without extensive ML expertise have a high demand for frameworks assisting people in applying the right ML algorithm to their problem setting. This is especially true in the field of smart energy system applications where more and more ML algorithms are used e.g. for time series forecasting. Generally, two groups of non-expert users are distinguished to perform energy time series forecasting. The first one includes the users who are familiar with statistics and ML but are not able to write the necessary programming code for training and evaluating ML models using the well-known trial-and-error approach. Such an approach is time consuming and wastes resources for constructing multiple models. The second group is even more inexperienced in programming and not knowledgeable in statistics and ML but wants to apply given ML solutions to their problem settings.

The goal of this thesis is to scientifically explore, in the context of more concrete use cases in the energy domain, how such non-expert users can be optimally supported in creating and performing ML tasks in practice on cluster computing environments. To support the first group of non-expert users, an easy-to-use modular extendable microservice-based ML solution for instrumenting and evaluating ML algorithms on top of a Big Data technology stack is conceptualized and evaluated. Our proposed solution facilitates applying trial-and-error approach by hiding the low level complexities from the users and introduces the best conditions to efficiently perform ML tasks in cluster computing environments.

To support the second group of non-expert users, the first solution is extended to realize meta learning approaches for automated model selection. We evaluate how meta learning technology can be efficiently applied to the problem space of data analytics for smart energy systems to assist energy system experts which are not data analytics experts in applying the right ML algorithms to their data analytics problems. To enhance the predictive performance of meta learning, an efficient characterization of energy time series datasets is required. To this end, Descriptive Statistics Time based Meta Features (DSTMF), a new kind of meta features, is designed to accurately capture the deep characteristics of energy time series datasets. We find that DSTMF outperforms the other state-of-the-art meta feature sets introduced in the literature to characterize energy time series datasets in terms of the accuracy of meta learning models and the time needed to extract them.

Further enhancement in the predictive performance of the meta learning classification model is achieved by training the meta learner on new efficient meta examples. To this end, we proposed two new approaches to generate new energy time series datasets to be used as training meta examples by the meta learner depending on the type of time series dataset (i.e. generation or energy consumption time series). We find that extending the original training sets with new meta examples generated by our approaches outperformed the case in which the original is extended by new simulated energy time series datasets.

Zusammenfassung

Die manuelle Entwicklung von Problemlösungen unter Nutzung von maschinellem Lernen (ML) ist selbst für erfahrene Spezialisten ein mühsamer und zeitaufwändiger Prozess, der viele Herausforderungen mit sich bringt, insbesondere, wenn man aus Skalierungsgründen Big Data ML-Frameworks einsetzen möchte, die auf leistungsfähigen Rechenclustern ausgeführt werden können. Durch das wachsende Interesse an ML in den letzten Jahren besteht auch bei Personen wie z.B. Ingenieuren ohne umfassende ML-Kenntnisse eine große Nachfrage nach Frameworks, die ihnen bei der Anwendung des richtigen ML-Algorithmus auf ihre Problemstellung helfen. Dies gilt u.a. auch für Anwendungen im Bereich intelligenter Energiesysteme, wo immer mehr ML-Algorithmen eingesetzt werden, z. B. für die Vorhersage von Zeitreihen oder die Zustandsabschätzung.

Ziel dieser Arbeit war es, zu evaluieren, wie Meta-Learning-Technologie effizient auf den Problemraum der Datenanalyse für intelligente Energiesysteme angewendet werden kann, um Energiesystem-Experten, die keine Datenanalyse-Experten sind, bei der Anwendung der richtigen ML-Algorithmen auf ihre Datenanalyse-Probleme zu unterstützen. Die Arbeit wurde im Rahmen der Forschungsarbeiten zur Konzeption und Implementierung einer digitalen Forschungsplattform für deutsche Energieforscher innerhalb des Helmholtz-Forschungsprogramms ESD durchgeführt. Daher sollten die Ergebnisse dieser Arbeit nahtlos als erweiterbare Lösung in diese größere digitale Forschungsplattform integriert werden können. Für die Evaluation und Verifikation der erarbeiteten wissenschaftlichen Ergebnisse wurden wesentliche Bausteine einer solchen integrierbaren Lösung unter Nutzung der entwickelten Methoden konzipiert und als Teile einer selbst-konzipierten und entwickelten Evaluierungsplattform prototypisch implementiert: Diese enthält u.a. ein einfach zu bedienendes, modular erweiterbares und auf einer Microservice-Architektur basierendes Framework für die Instrumentierung, Nutzung und Evaluierung von ML-Algorithmen unter Verwendung von Open Source Big Data ML Software Stacks, welches auf Computer-Clustern lauffähig ist und über eine modulare, webbasierte Benutzeroberfläche ohne große Kenntnisse der Laufzeitumgebung genutzt werden kann, ein generisches Meta-Learner-Framework, das diese Umgebung so erweitert, dass Verfahren zur automatischen ML-Modellselektion basierend auf Meta Learning getestet und evaluiert werden können sowie ein generisches Konzept, wie Meta-Features durch Meta-Feature-Sets zunächst unabhängig von einer gegebenen Aufgabenstellung im Framework repräsentiert werden können.. Des weiteren enthält das Framework Werkzeuge für automatisiertes Datenmanagement, zur Vorverarbeitung und Bereinigung Zeitreihen-basierten Datensätzen für Trainings- und Evaluationszwecke sowie zur Erzeugung, zum Management und zur Pflege von Test- und Trainingsdatensätzen für das Meta Learning.

Um die wissenschaftlichen Fragestellungen zur Nutzbarkeit von Meta Learning zur Unterstützung von Nicht-ML-Experten bei der Auswahl und Instrumentierung von ML-Algorithmen mit Hilfe dieses Frameworks evaluieren zu können, wurden konkrete Anwendungsfälle auf der Evaluierungsplattform instrumentiert, wobei der Schwerpunkt aus pragmatischen Gründen (Verfügbarkeit von Daten, verfügbare Zeit) auf der Durchführung von Zeitreihenanalysen und Prognosen für energie-bezogene Last- und Erzeugungszeitreihen mit Hilfe dafür geeigneter ML-Modelle lag. Zur Instrumentierung der Algorithmen-Selektion für diese Anwendungsfälle wurde des Weiteren ein neuer dedizierter Satz von Meta-Merkmalen (DSTMF) für solche Zeitreihenanalysen zur Algorithmen-Selektion konzipiert und mit anderen bekannten Meta Feature Sets verglichen. Unter Nutzung der instrumentierten Anwendungsfälle wurden dann auf der Evaluierungsplattform zahlreiche Datenanalyse-Experimente durchgeführt, um die wesentlichen Aspekte der Meta-Lernplattform quantitativ zu erfassen. Diese Evaluationsdaten wurden dann analysiert, um die Hauptaussagen der in der Einleitung dieser Arbeit vorgestellten Forschungsfragen zu verifizieren.

Dabei konnte gezeigt werden, dass das über eine Weboberfläche bedienbare selbst-konzipierte Microservice-basierte Framework zum Management und zur Ausführung von ML-Jobs selbst ML-Experten bereits ein spürbare Erleichterung in der Nutzung von Big Data ML-Berechnungsumgebungen, welche auf Cluster-Computing Umgebungen laufen, bieten kann, da sich in eine solche Umgebung einer Reihe weiterer Werkzeuge z.B. zur Erzeugung und zum Management von Test- und Trainingsdatensätzen (hier wurden verschiedene Lösungsansätze für die Generierung von Energiezeitreihen für Test- und Trainingszwecke entwickelt und evaluiert), zur automatischen Parametrisierung eines Algorithmus auf der Cluster-Computing Umgebung, zum Caching bereits trainierter Modelle sowie zur Anzeige und Erfassung von Laufzeit- und Performanzdaten von Modellen, etc. integrieren lassen, welche den ML-Experten bei der Evaluation, Parametrierung und Bewertung neuer Algorithmen auf Cluster-Computing-Umgebung essentiell unterstützen. Dabei konnte auch gezeigt werden, dass solche Werkzeuge die Laufzeitperformanz der Algorithmen kaum beeinträchtigen sondern ab einer gewissen Komplexität des Lösungsraums drastisch erhöhen, was wiederum den Entwicklungsaufwand solcher Lösungen reduziert. Die gesamte Umgebung kann dabei hochgradig skalierbar, modular und erweiterbar konzipiert werden, und lässt sich damit problemlos in andere Microservice basierte Umgebungen wie die Helmholtz-Plattform zur Energieforschung integrieren.

Des Weiteren konnte gezeigt werden, dass die Integration einer generischen Meta Learning Lösung eine sehr einfache Nutzbarkeit bereits in der Plattform implementierter ML-Algorithmen durch Nicht-ML-Experten ermöglicht. Eine solche Meta-Learning Lösung lässt sich weitgehend generisch von der Methodik her realisieren, wie dies im Rahmen der Arbeit durch Entwicklung eines eigenen methodischen Ansatzes hierfür auch gezeigt wurde. Allerdings konnte in der Evaluation unter Nutzung des DSTMF-Meta-Feature Satzes auch gezeigt werden, dass eine solche Methodik für jede Anwendungsproblemklasse einen spezifischen Satz von Meta-Feature-Attributen erfordert, damit die Algorithmen-Selektion für diese Anwendungsklasse auch optimal durchgeführt wird. Bei der Realisierung von Anwendungsframeworks ist es daher wichtig, dass sich solche Meta-Feature-Sets für neue Problemklassen problemlos in eine bereits bestehende Framework-Umgebung integrieren

lassen, wobei der Meta Learner anschließend dann auch für die neue Aufgabenstellung neu antrainiert und hierfür wiederum geeignete Trainingsdaten verwaltet, erzeugt und bereitgestellt werden müssen, was dann wiederum Aufgabe von ML-Experten ist. Zur Unterstützung dieser können dann wiederum die bereits oben erwähnten Ansätze genutzt werden.

Im Fazit kann gesagt werden, dass die in der Arbeit entwickelten und beschrieben Ansätze die Realisierung kommerzieller oder freier ML-Umgebungen zum Einsatz auf Cluster-Computing-Umgebungen On-Premise oder in der Cloud ermöglichen, welche die Nutzung von ML-Lösungen im kommerziellen Umfeld für Nutzer wesentlich erleichtern. Allerdings ist hierzu noch einiger Entwicklungsaufwand nötig.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Machine Learning (ML) is a scientific discipline aiming at designing and developing specific algorithms and concepts allowing computers to evolve behaviors and react to different actions based on empirical data. Indeed, it can be seen as a core in the field of artificial intelligence, in which computers can learn from existing data to predict future behavior, results and trends. Over the last decade, ML has been applied in a lot of problem fields, such as text classification [139], speech recognition [109], medical diagnostics [126], computer vision [68] and computer graphics [3].

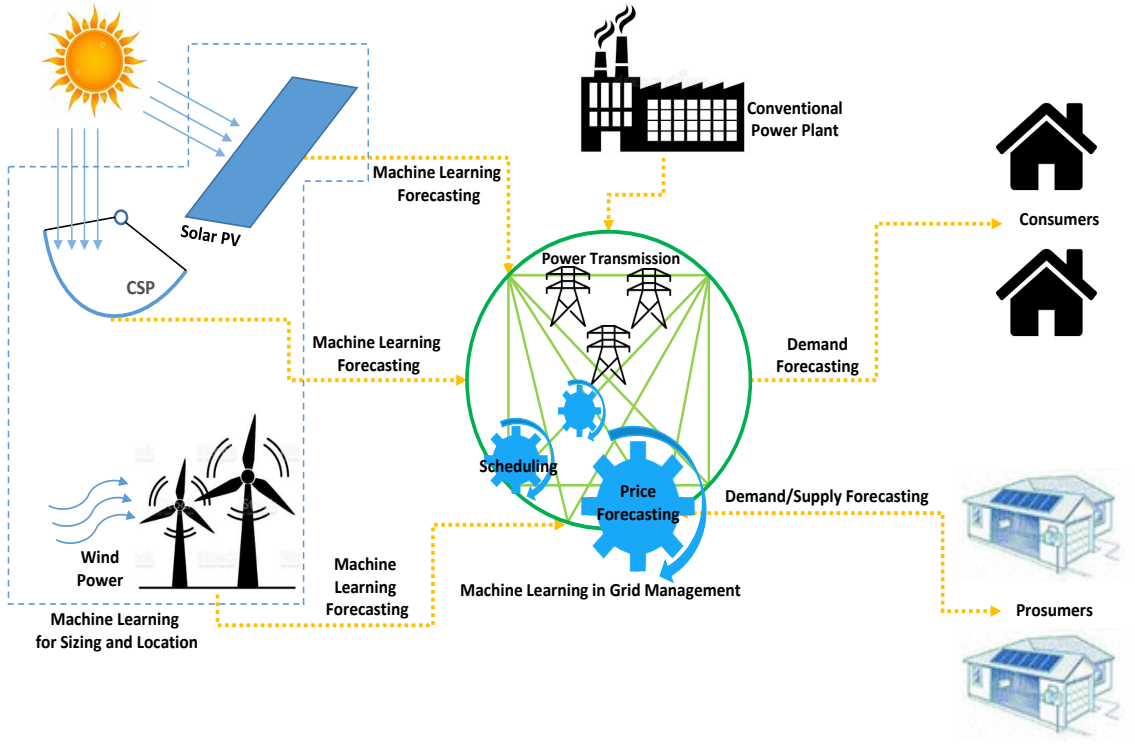


Figure 1.1.: Overview of power grid with integrated renewable sources and its usage of machine learning techniques in different steps [114].

In the field of energy, ML has also been successfully applied [12][179][43][1][178][38][170]. One usage area in this application domain is to use ML algorithms for intelligent decision making in unit commitment of decentralized renewable energy resources and flexible loads

at grid level, where e.g. an accurate prediction of future energy demand and renewable energy generation is required as seen in Figure 1.1. Such predictions are prerequisites for optimizing the usage and minimizing wastage of energy in the system, paving the road for a more efficient usage of power networks at the grid level.

To support the German Energiewende aiming at maximizing the usage of renewable energy resources, and thereby reducing non-renewable energy production, the German government funded several larger scale energy labs, e.g. the Energy Lab 2.0 at the KIT or the Living Lab Energy Campus (LLEC) at FZJ, which are used in context of the Helmholtz research programme Energy Systems Design (ESD) as environments for research on new smart digital solutions for controlling and managing future energy networks. Instrumenting and evaluating new digital system solutions for controlling hybrid energy systems combining different technologies from different sectors is a very complex task requiring an adequate software ecosystem for e.g. gathering data and performing data analytics, modelling and executing control algorithms in experimental settings. Therefore, one main research subtopic of the ESD programme focuses on the development of a new digital energy research platform which easens the work of the energy researchers by providing a ready to go integrated tool ecosystem that hides the IT-related complexities of data gathering and managing, data analytics, managing and executing control algorithms and simulation models on high performance computing clusters from the users, which can then focus on their own research questions. This digital platform should also provide easy solutions for applying and evaluating the use of existing and new ML algorithms as part of new smart grid control solutions, and one important sub-case is the application and evaluation of adequate ML algorithms for forecasting time series data of renewable energy production and/or volatile load which are then used as part of more complex control algorithms for e.g. unit commitment of energy resources.

Table 1.1.: User categories.

Category Nr.	User category		Properties
1	Expert		ML knowledge (+) ML Programming skills (+)
2	Non-expert	**A**	ML knowledge (+) ML Programming skills (-)
		B	ML knowledge (-) ML Programming skills (-)

The users performing such ML tasks on the digital research platform can be divided into two different groups, namely experts and non-experts as shown in Table 1.1. On the one hand, the expert users have a deep understanding of ML and also good programming skills to implement ML models using, for example, some developing tools like Jupiter Notebook [116].They have worked with ML libraries before and are capable of programming algorithms themselves. On the other hand, many energy researchers are non-experts in programming ML algorithms or even in understanding different ML algorithms. They

are only interested in applying ML algorithms to their problem settings. Therefore, such non-expert ML users of the platform are grouped into two sub-categories.

The first one includes the users who are familiar with statistics and ML but are not able or interested in writing the necessary programming code for training and evaluating ML models particularly in cluster computing or Big Data environments. The second sub-category of non-expert users has only little to no knowledge about statistics and ML. They only want to apply ML algorithms to their research problems. The general research question for the digital energy research platform with respect to ML usage can now be formulated as "How can we support the two different Non-expert user groups mentioned before in applying ML algorithms to their research setting without forcing them to become all ML experts with detailed knowledge in ML methodology and programming?". This thesis will try to answer this question while concentrating on a smaller group of use cases related to time series forecasting for experimentally evaluating the basic ideas for solving the above problem.

Generally, the process of building ML models, which can then later be applied to a given problem setting, consists of multiple steps and is commonly called Machine Learning Pipeline (MLP). Figure 1.2 shows a simplified MLP encompassing data preprocessing, splitting the data into training and test data, model training and model testing.

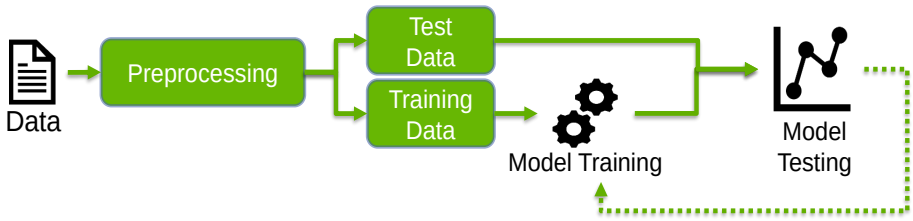


Figure 1.2.: Simplified Machine Learning Pipeline (MLP).

For training accurate models for more complex problem settings, often the size and diversity of the dataset used for training the model is very important. Using large scale datasets for training, testing, and executing MLP effectively is becoming more difficult and even complex, requiring the utilization of computing clusters with dedicated ML software designed for parallel execution of parts of the algorithm (e.g ML tools designed for cluster computing environments or so called Big Data Platforms [50]). However, gaining insightful information, finding patterns and extracting knowledge from such datasets are quite complex tasks for ML users too. Additionally, the right cluster configuration of the MLP pipeline on the cluster computing infrastructure beside setting up the the required communication environments to send jobs to the cluster introduce more challenges for running ML tasks for non-experts, who otherwise must know how parallel execution of ML frameworks on a cluster computing environment can be optimized for a certain problem setting.

Building on that, more in-depth research on new methodologies and an even-growing software solutions for facilitating performing ML tasks for the aforementioned ML users are existing, see e.g. [26][60][33][185][27][166]. Such solutions can be categorized into

monolithic single user applications and service-oriented solutions often having web user interfaces and/or command line interfaces (CLI) as user interface frontend. Newer service-oriented solutions nowadays often use a microservice based architecture [30]. In contrast to monolithic applications, an application with a microservice architecture is composed out of several independent deployable services, where each individual microservice performs a specific task based on its own technology stack [19]. Microservices represent state-of-the-art technology, where each service is designed to be horizontally scalable in order to build highly modular, flexible and scalable solutions. Additionally, microservice applications are designed to be automatically deployable and manageable on computing clusters providing a container runtime environment (e.g. Docker) and/or container orchestration software (e.g. Kubernetes) which is quite common in cluster computing environments nowadays. They can be easily integrated with other service-oriented applications, e.g. distributed data analytics or data management environments which nowadays also run seamlessly on computing clusters providing a container runtime environment. Thus, such applications can be easily integrated with modern Big Data data analytics frameworks, e.g. Apache Spark for parallel execution of classical machine learning algorithms or TensorFlow for executing Deep Learning Algorithms in the same cluster computing environment.

However, most of the existing solutions to support non-expert users in performing ML tasks were developed in the past as monolithic applications and often also do not support executing ML jobs in cluster computing environments. These types of applications are often tuned to allow users to easily perform certain ML tasks, e.g. classification, and provide a great user experience. But they are not suitable when the problems get more complex, the datasets for training will become too large for single computer environments, and therefore the MLP has to be performed on cluster computing environments. Driven by that, the research on modern energy system solutions must nowaday focus on more complex real world usage scenarios for making the German Energiewende a reality. And the evaluation of real world usage scenarios demand very complex system models and often large scale datasets for e.g. train ML based data analytics algorithm as part of a solution which can only be executed on computer cluster environments. Therefore, the working group designing the architecture of the digital energy research platform for the ESD research program of Helmholtz has decided that this platform will be conceptualized and implemented using a microservice-based architecture, where all services will run on Kubernetes based cluster computing environments and the user interfaces will be implemented as (progressive) web applications which can be used from any frontend device (e.g. desktop computer, laptop, tablet or smartphone). For executing MLP and ML tasks on this microservice based platform, the first challenge of this thesis can be formulated as “designing a Microservice based solution for supporting expert users and non-expert users, category 2A, in performing ML tasks in a cluster computing environment with container runtime automation”.

Another problem is, that according to the No Free Lunch (NFL) theorem [173] no single learning algorithm has always the lowest performance error on a broad problem domain. Hence, for solving a specific subclass of problems of this broad problem domain, a dedicated learning algorithm must be selected from a set of available solutions with lowest performance error. This selection process can be defined as an Algorithm Selection Prob-

lem (ASP) [125], which is typically solved by the ML user using a trial-and-error approach. I.e. all possible combinations of learning algorithms with their hyperparameters are tried to find the most suitable solution. This can take a long time even when using a cluster computing environment, and has a high computational complexity due to the size of the search space of possible algorithm candidates. Moreover, expert knowledge is required to perform such tasks. Driven by that, the second challenge of the thesis can be formulated in conceptualizing a methodological approach which allows the digital energy research platform to support non-expert users, category 2B, in selecting the most appropriate ML model for a specific energy research related task.

According to [81][151][25][70][122][127][45][128], the usage of meta learning approaches for solving this problem is not new. They have been proposed as good solutions to deal with the problem of algorithm selection supporting non-expert users shown in Table 1.1 for automatically finding the most appropriate model in scientific literature for a long time. But the challenge for this thesis is not to decide if meta learning in general can be used to solve the problem, but more "how meta learning has to be instrumented in the field of energy for successfully solving the algorithm selection problem". The term meta learning is based on the fact that a good set of characteristics of datasets captures powerful insights into the behavior of these datasets paving the road for determining the most appropriate ML algorithm [16][55]. But these characteristics of the datasets largely depend on the characteristics of the system environment where the data was gathered, and therefore also on the application domain. Consequently, the better the used set of characterization indicators captures the behavior of a dataset with respect to their dependency on the domain specific system environment, the more accurate the selection of the best ML algorithm will be. Therefore, the main challenge for instrumenting meta learning for the digital energy research platform is finding energy system environment specific metadata sets for different ML tasks.

However, providing a generic meta learning machinery, which can be adapted by configuration and recurring learning to solve the algorithm selection problem for different ML tasks, and enhancing the predictive performance of the meta learning classification model to perform well in a cluster computing environment still represents a challenge that needs to be faced.

1.2. Research Questions and Contributions

This thesis has the main goal of supporting non-expert users, presented in Table 1.1, in performing ML tasks in cluster computing environments using e.g. Big Data analytic tools. Energy load and generation forecasting scenarios are used to experimentally evaluate the efficiency of the proposed solution. The first group of non-expert users are supported by developing a conceptual microservice-based framework with highly configurable web-based UI hiding the low level complexities of the computing cluster infrastructure and Big Data analytic tools running on the cluster from the users. Such framework can be seamlessly integrated into the digital energy research platform of Helmholtz. The second

group of non-expert users are supported by extending the microservice-based solution to incorporate a generic meta learning solution. This solution is specifically designed for solving the algorithm selection problem for energy related ML tasks such as energy time series forecasting. Additionally, scientific questions of what are the good meta features for e.g. solving the algorithm selection problem for the ML task forecasting of energy time series and how can the performance of the algorithm selection process be enhanced by enhancing the training datasets by generating new datasets are addressed. Four research questions are formulated to cover the main goals of the thesis. The following description of these research questions also lists the scientific contributions which were worked out during the thesis to find the answers of these questions and address the motivation presented in the previous section.

Research Question 1 [RQ1]: How should non-expert users with ML knowledge, but without programming skills be supported in performing ML tasks on computing clusters using Big Data ML frameworks?

The rapid extension of successful large scale ML applications and the immense growth of available data usable for training ML solutions aroused the interest of other developers and researchers for developing solutions using ML based Big Data analytic tools on cluster computing environments for solving complex data analytic problems. But programming ML solutions using Big Data ML tools is challenging for non-expert users who do not have programming skills in ML and modern Big Data technologies. In addition, a distributed execution environment is required for training such solutions in a timely manner on a big training dataset.

Supporting non-expert users, category 2A presented in Table 1.1, in performing ML tasks by providing easy-to-use tools is nothing new. Such users have a deep understanding of ML and its related learning scenarios but they are not able to write the programming code required to perform ML tasks as already discussed before. To support them, there have been a lot of research projects [23][26][27] which developed dedicated tools implementing certain classes of ML algorithms (e.g. classification or regression) in a very generic fashion that allows category 2A users to apply those algorithms to their problem setting without the need for programming skills. Most of those tools are developed as monolithic applications which have drawbacks in terms of scalability and maintainability as discussed before. The majority of them do not have the ability to scale beyond a level where the computing power or the data management capability of a single computer is not enough for coping with either the computational complexity of the problem or the amount of training data. There are some tools (e.g. Jupiter Notebook) found in literature, which use distributed ML frameworks for executing more complex ML problems on cluster computing environments using distributed parallel computation algorithms which also allow to distribute needed data e.g. for learning onto different nodes of the cluster. Such tools are often called Big Data analytics tools (e.g. Apache Spark for classical ML algorithms or TensorFlow for Deep Learning). But these frameworks do not hide the low level complexities of the used Big Data framework and require that the user should have skills and knowledge in both ML and the structure and working of the underlying framework on a computing cluster [33].

In this thesis, we tackle the aforementioned challenges and answer the scientific question of designing an efficient adequate solution for supporting non-expert users by proposing a microservice-based framework which hides the complexities of the big data runtime environment from the users allowing them to perform their tasks without caring too much about the technical issues of the underlying big data platform. The solution can be set up on any computing cluster providing a container runtime environment for efficient distributed processing of MLPs by instrumenting already available Big Data ML frameworks and hiding their usage from the ML users. A modern web-based user interface supports non-expert users in performing ML tasks from anywhere with any device providing a web browser. This framework can be integrated with the digital energy platform for ESD. E.g. datasets stored in that platform can be easily used for training of ML algorithms or ML algorithms can be directly applied to such datasets.

The scientific work to answer this research question is detailed in Chapter 4. The following bullet points summarize the main scientific contributions presented in this chapter:

- Conceptualization of a microservice-based framework to support non-expert users category 2A in training, testing, managing, storing, and retrieving ML models using Big Data ML frameworks installed on computing clusters with a container automation runtime environment.
- A benchmark evaluation study was carried out for determining the overhead of such a framework by prototypically implementing the microservice framework and performing ML tasks instrumenting the use case of energy load time series forecasting using classic ML algorithms already implemented in the Big Data ML framework Apache Spark. In this study, the effect of instrumenting the caching of data in Resilient Distributed Datasets (RDDs) in Apache Spark on the performance of the forecasting algorithm was also evaluated. The efficiency of the framework in terms of execution time and overhead is also measured focusing on the effect of storing and retrieving ML models.
- Defining and evaluating the thresholds, at which it is highly recommended to switch to a Big Data clustering infrastructure for time series forecasting because it outperforms the usage of the same ML frameworks in single computer context.

After conducting this work for supporting the first category of non-expert users, the focus is moved to conceptualizing tools for supporting the second category of non-expert users. After an extensive literature study (see Chapter 3), the instrumentation of a meta learning solution for automating the process of model selection seemed to be very feasible. To conceptualize and evaluate this approach,two major aspects and challenges need to be addressed in this thesis. On the one hand, the meta features that adequately describe the input datasets are largely dependent on the system context in which the datasets belong and on the specific ML task which should be performed. Therefore, for evaluating the meta learning approach by instrumenting the concrete use case forecasting time series datasets with ML algorithms, adequate meta features need to be found which give a good prediction performance for finding the right forecasting ML algorithm. On the other hand, the prediction performance also depends on the number and availability of good training

meta examples on which the meta learner is trained to recommend the best model for a specific task. These challenges turned out to be complex research problems by themself which lead to the second and third research question.

Research Question 2 [RQ2]: How to efficiently characterize energy time series datasets to enhance the performance of automated model selection?

For evaluating our meta learning approach in the context of energy, time series data forecasting in energy context is used as a use case. In this approach, the meta learner has to learn to automatically assign the best ML forecasting model to an energy time series dataset [38][91]. This is normally done by learning the mapping between the forecasting performance of ML algorithms and the characteristics of energy time series datasets to forecast. Meta learning leverages the capability of ML classification algorithms to find such relationships and to classify input energy time series datasets according to ML candidates, e.g. determining an adequate forecasting model [118]. Changing the characteristics of energy time series datasets may affect the assigning process of the meta learner [16].

This raises the question, how to efficiently characterize energy time series datasets to precisely assign the best forecasting model for them. In meta learning, the characteristics of datasets are called meta features. They define a set of features (properties) derived from a dataset which e.g. cover statistical properties describing the time-related behavior of an energy time series dataset. Describing characteristics of energy time series datasets using such meta features is not new. There has been a lot of research in this context [91][151][29]. In addition to the high computation time for most of them, especially for large amounts of energy time series datasets, the calculation of these meta features is often subject to privacy and security issues. E.g., the meta features that are calculated by using knowledge about the physical surroundings (e.g. a certain building) or about human behavior of people in households. Therefore, those meta features are not optimal for using them for meta learning because they would need to be applied on all training data sets. Consequently, it is important for setting up the evaluation use case for the meta learning framework to design a new set of meta features that indirectly and anonymously captures the physical and other social properties in energy time series datasets with an acceptable as well as efficient extraction time.

The scientific work done for designing and evaluating a more suitable set of meta features for time series forecasting is described in Chapter 5. The main scientific contributions can be summarized as follows:

- Introduction of Descriptive Statistics Time-based Meta Features (DSTMF) as a new set of meta features for characterizing the forecasting behavior of time series datasets to achieve a more accurate and more performant model selection for time series energy load forecasting.
- A similarity-based clustering analysis study was carried out to evaluate the potential of DSTMF's meta features for capturing the deep characteristics of energy load time series datasets in comparison to other state-of-the-art meta features used in the field of energy.

- To further assess the effectiveness of DSTMF in characterizing energy time series datasets, a generic Energy Meta Learning System (EMLS) was conceptualized and implemented. EMLS allows to perform meta learning with different sets of meta features for comparison. Using that, an in-depth evaluation study was performed whereby the predictive performance of our meta learning classification model using DSTMF was compared to the predictive performance using other state-of-the-art meta features from the literature.
- The EMLS system was further enhanced by designing the Encoded Energy Meta Learning System (EEMLS) leveraging the advantage of unsupervised deep learning to encode the extracted meta features by autoencoders. The resulting meta learning classification model achieves a very good predictive performance with an average accuracy value of 90%. This accuracy is achieved even when using a reduced number of training examples.

Research Question 3 [RQ3]: How to enhance the performance of automated model selection in the context of energy by creating appropriate learning datasets?

Multiclass classification problems aim at assigning a class label from a set of classes to each input example. It categorizes the input data according to different classes. Driven by that, the well-known Algorithm Selection Problem (ASP) is considered as multiclass classification problem, where an input dataset needs to be assigned an appropriate algorithm from different ones available in the class space of algorithms. Typically, the predictive accuracy of such a classification model not only depends on the features used for classification but also on the availability of a bigger set of diverse examples required for training.

Similarly, meta learning as a solution of ASP is defined as a multiclass classification problem [76][91]. The reason for that lies in the fact that the meta learning classification model captures the mapping between meta features, that describe the data, and the predictive performance of the best model. As a result, the output of meta learning will be assigning each input dataset into the best ML model from different model candidates. In addition to the challenge of availability of efficient meta features that has been tackled in the previous scientific question, the availability of examples required for training meta learning classifiers represents another challenge that needs to be faced. An appropriate diverse training dataset containing time series datasets with different temporal behavior plays a crucial role in supporting the meta learning classifier to efficiently learn the mapping between meta features and the best forecasting model. But because of security and privacy constraints, there aren't too many load and generation time series datasets available to the general public.

This led to much research work focusing on generating new synthetic time series datasets for increasing the number of training examples required for the generalization process of classifier [47][156][121][65][188][64]. In the present thesis, we answer the scientific question of enhancing the predictive performance of ASP in the context of energy by conceptualizing and developing new approaches for generating new adequate time series datasets for the energy specific application domain for better training of our meta learner. The advantages of our approaches over existing ones in literature lies in the simplicity of

the training dataset generation approach, which nevertheless achieves a high predictive performance of the meta learning classification model in assigning the best forecasting model for input energy time series datasets.

More details on the scientific work done for answering this research question can be found in Chapter 6. The following bullet points summarize the main scientific contributions presented in this chapter:

- Developing and conceptualizing a weather-based approach for generating new energy time series datasets for i.e. generation scenarios leveraging the advantage of weather data and its weather conditions. Such conditions are used to classify the input energy time series dataset into different new ones.
- Developing and conceptualizing an aggregation-based approach for generating new energy time series datasets by aggregating the input time series into different granularity levels.
- A benchmark evaluation study was carried out using a power generation forecasting scenario to evaluate the efficiency of the new generating time series datasets. In this evaluation, the new approaches are compared to the model-based approach existing in the literature for enhancing the predictive performance of the meta learning classification model. Moreover, the advantage of unsupervised deep learning was incorporated into meta learning for achieving a better performance in energy time series model selection.

Research Question 4 [RQ4]: How should non-expert users with neither ML knowledge, nor programming skills be supported in performing energy time series forecasting in Big Data environments?

Meta learning in the field of energy is closely linked to the process of finding the mapping between meta knowledge that describes the energy time series datasets and the best performing forecasting models. Having such a mapping, the Algorithm Selection Problem (ASP) can be solved by recommending the best forecasting model based on the characteristics of energy input energy time series datasets without the need to follow a trial-and-error approach. But how can we instrument such a meta learning solution in a data analytics platform for the energy engineering domain for users in category 2B which can be integrated easily into the digital energy research platform developed in the context of ESD research?

In some research projects, meta learning approaches were developed as frameworks with wizard-like interfaces which are very easy to use for the non-expert [158][75][45][87]. Despite the promising advances achieved in this context, much work remains to be done. None of those existing frameworks were developed with a software architecture - such as a microservice-based architecture that allows to easily integrate the meta learner into a bigger digital energy research platform, nevertheless providing a good maintainability as well as efficiency of the system and scalability for future use. Moreover, these easy-to-use meta learner tools typically do not use ML Big Data analytic frameworks for solving the ASP on cluster computing environments for higher performance in the case of Big Data.

The scientific work done to answer the fourth research question brought together the results of the aforementioned research questions to set up a very flexible data analytics framework which can be easily integrated into the digital energy research platform of the ESD project. It merges the concepts of meta learning addressed in RQ2 and RQ3 and the microservice-based solution addressed in RQ1 as a microservice-based meta learning framework in Big Data environments to automatically select the best forecasting model for non-expert users of category 2B presented in Table 1.1.

By answering this research question in Chapter 7, the following scientific contributions are summarized:

- Conceptualization and development of a microservice-based meta learning framework solution to automate the process of Algorithm Selection Problem (ASP) in the energy field which is easily integratable into the digital energy research platform of the ESD research program. The proposed framework makes the use of a microservice architecture built on top of a powerful big data stack for a manageable and highly scalable solution to solve the problem of model selection in big data environments.
- A benchmark evaluation study was carried out to evaluate the accuracy of the meta learner, the execution time and the overhead of the framework.

1.3. Structure of the Thesis

According to the research questions presented before, this thesis is structured as follows. In Chapter 2, we present the basic background knowledge necessary to understand the subsequent parts of this thesis. Chapter 3 summarizes a variety of previous research projects that suggest scientific methods and frameworks related to our work. We discuss the strengths and weaknesses of each project and clarify its relation to our work. Chapter 4 covers the research question RQ1 by developing an adequate runtime environment to support non-expert users, category 2A in performing energy forecasting in the context of big data. Besides the execution workflow and the conceptual architecture including its services and the main functionalities, an in-depth evaluation study concerning the performance of the proposed framework in terms of execution time and overhead is presented in this chapter. In chapter 5, we further answer the research question RQ2 in presenting Descriptive Statistics Time based Meta Features (DSTMF) as a new form of meta features to deeply characterize energy time series datasets with respect to forecasting. A benchmark evaluation study for evaluating the performance of DSTMF compared to the performance of other state-of-the-art meta features existing in literature in the field of energy is presented in this chapter. Chapter 6 covers the research question RQ3 by enhancing the predictive performance of the meta learning classification model by generating new meta examples related to the input energy time series datasets. In chapter 7, we address research Question RQ4 by integrating the meta learning concepts introduced in Chapter 5 and 6 into the overall ML microservice framework defined in Chapter 4 and providing an easy-to-use wizard-like interfaces to assist non-expert users. The chapter

focuses on the conceptual microservice-based architecture proposed to support non-expert users, category 2B, in selecting the most appropriate forecasting model by meta learning. Finally, in Chapter 8, the main findings of the thesis are summarized and an outlook on further work are given.

2. Theoretical Background

For better understanding of the main contributions presented in this work, it is of great importance to introduce some of the related theoretical background including fundamental terms and state-of-the-art techniques. The structure of the background is as follows. We start in Section 2.1 by explaining some fundamental concepts of Machine Learning (ML), its learning scenarios, application area and the different measurements proposed in the literature to evaluate the predictive performance of ML models. In Section 2.2, the basic concepts of Big Data analysis for handling the increasing amount of data generated everyday are explained. In addition, the microservice-based architecture style as a new technology for building efficient software solutions is introduced in Section 2.3. Such architecture is presented along with its characteristics, bounded contexts and the related service communication types. All this knowledge is necessary for understanding key concepts of this thesis [146].

2.1. Machine Learning

The ability to learn from already existing data and iteratively find a solution for performing certain tasks by adjusting the application's behavior according to knowledge gathered from the data is the main goal of ML. An essential description of ML is provided by Alpaydin et. al [10]:

"Optimizing a performance criterion using example data and past experience".

Using machine learning, the computers are able to learn from historical data (experience) and use the acquired knowledge to efficiently perform further tasks. Another more formal definition of ML is given by Jordan et. al in [61]:

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E".

In this definition, the "computer program" termed "machine learning model" can be seen as an approximate solution of some complex mathematical function (model) which for some given input variables calculates a set of outputs for performing the given task T (e.g., a classification). By learning from experience E (i.e., from some data which contains

information about the correct function values for specific input variables settings) the internal configuration of the ML model will be subsequently changed and thereby optimized in a learning phase such that the model better performs its approximation. ML has already been proven in many application areas to be able to provide very good approximate solutions for problems which are otherwise too complex to e.g. set up differential equations describing the problem environment with all its real world physical effects, and then solve the differential equations appropriately. To this end, applying ML to extract useful knowledge from raw data has become increasingly popular in a variety of areas. One such field is the health sector where it helps with medical diagnosis [37][126]. Virtual voice assistance, like Siri and Alexa, is another example, where ML is used to determine the meaning of voice commands from people like setting the alarm clock or finding specific information on the internet.

ML is nowadays also used for solving complex problems in technical industrial application areas, such as smart energy systems. This thesis will later focus on facilitating the usage of ML for energy load and generation forecasting [12][179][43][1][38][178][170]. Text classification [62] is a further example of an application of ML which can be used to classify documents according to more than one category such as sport, medicine, and health care, to name a few. Moreover, the extraordinary increase in the size of multimedia data led to an increasing interest in applying ML as a potential way to provide the opportunity to index, restore as well as to browse such data via keywords semantically [174][119][160]. The field of bioinformatics represents another important application area of ML [80][147] where proteins and genes are studied and classified according to their functional classes. In chemical analysis [67], the major advancements in the pharmaceutical industry combined with new achievements in ML algorithms and learning strategies pave the road for discovering and developing multiple-action drugs. By using ML, it becomes possible to discover the drugs that have the ability to achieve several therapeutic goals simultaneously.

2.1.1. Machine Learning Scenarios

On top level, four different learning scenarios can be distinguished in the field of machine learning, namely supervised, unsupervised, semi-supervised, and reinforcement machine learning scenarios. Those can be further divided into sub-types. The main distinction between the mentioned scenarios depends on how the ML system knows and therefore the ML model learns that a certain solution defines a correct answer to the problem for a given set of input variables.

Supervised Machine Learning: In this scenario, the learning data (i.e. well known sets of input variable values) are labeled with significant information called labels [7][150][107]. The learning dataset is then split into two parts. The training dataset and the test dataset. The training dataset is then used to learn the existing mapping between input values (predictors) and labels. For evaluation, if the trained model performs a good approximation of this mapping, the test dataset will be used to predict the output values (and therefore the label values of test data sets) of each test dataset. Then the prediction will be compared with the labels, and an error for the deviation will be calculated (see later). The error

will then be used to decide how to proceed with the learning. Both predictors and their corresponding outputs could be nominal or numeric depending on the source of data. In the supervised learning settings, we can think of a teacher which provides extra information i.e., labels to the examples in the training dataset to e.g. give hints on how to predict such information for the unlabeled examples in the testing dataset.

In the supervised learning scenario, the learning process involves finding the optimal mathematical mapping between the input predictors and output labels. In practice, often two well-known types of problems, namely classification and regression problems are solved using supervised machine learning [135]. In classification, the output are discrete values (i.e. class names of a classification system) whereas in regression, the output values are continuous numbers (e.g. from a subset of the real numbers for instance). A common classification problem is e.g. classifying whether an email is spam or not. In contrast, regression can be used for the prediction of the prices of houses in certain areas of a town, or for the prediction of the power generation of renewable energy sources.

When using supervised algorithms, the objective is to train models which generalize well on input data. However, depending on the actual problem setting, model generalization is often very difficult to achieve and related to it is the concept of the bias-variance tradeoff [22][181], where models have either high bias and low variance or low bias and high variance, and both characteristics cannot be optimised at the same time. Simple ML models tend to have high bias and low variance in contrast to the real world problems they should approximate, and therefore they are often unable to capture the complex underlying patterns of data. The result of this is called underfitting [56] [189], where the model performs poorly on the training data and is not able to generalise to new data. On the other hand, overfitting [130][183] is when a model learns too many parameters and gets too complex, resulting in a high variance but decreased bias. Overfitting is a common problem in supervised learning, especially when the samples are few and the feature dimensionality is high. Therefore, it is crucial that the number of data instances is higher than the independent parameters to lessen the likelihood of overfitting.

Unsupervised Machine Learning: In contrast to the aforementioned scenario, in which the examples are explicitly labeled, the examples here are unlabeled. There is no information in the training set except the features (input variable values) without the corresponding output [71][28]. In this context, the unsupervised machine learning process e.g. can try to discover the similar characteristics between groups of input data and/or a significant structure in data, for example, by grouping them into different meaningful clusters as seen in Figure 2.1. Some important application examples in the context of unsupervised machine learning scenario are:

- The k-means algorithm for clustering tasks [4][44].
- The Apriori algorithm for association rule learning tasks [90] [13]. This algorithm is mainly used in recommendation systems aiming at discovering the behavior of customers and presenting the appropriate products to them consequently.

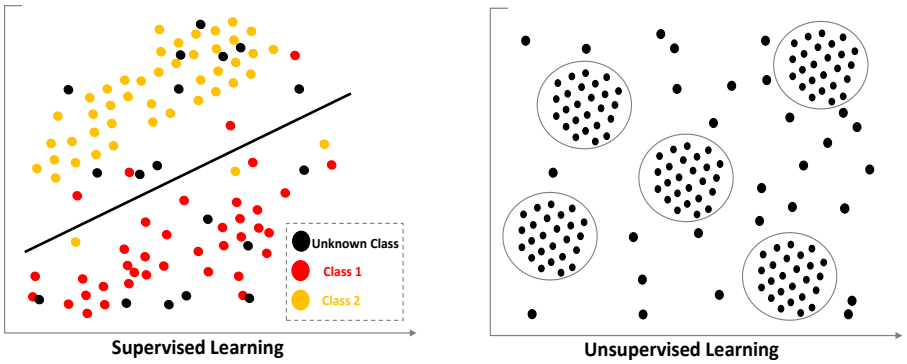


Figure 2.1.: Prediction of future energy demand and renewable energy generation.

Figure 2.1 shows the main differences between supervised and unsupervised learning scenarios. The unsupervised learning scenario (shown in the right picture) aims at discovering the structure of data in the context of exploratory analysis. Clustering is an example of unsupervised learning in which the data points are grouped together into different clusters based on their characteristics. The supervised learning scenario (see left picture) aims at classifying the data points in different classes based on some classification criteria. To achieve that, the learning algorithm learns the relationships between input data points and the output classes. In this learning process, each data point is mapped to its corresponding class to formulate the final classification distribution of points.

Semi-supervised Machine Learning: It can be seen as a mixed form of the two aforementioned learning scenarios, whereby only a part of the data is labeled with some supervision information i.e., labels [162][180][17]. Consequently, a semi-supervised machine learning scenario is often easier to instrument than the supervised one because labeling all data sets can be very difficult and time consuming. It is often difficult to get labeled data because the annotation of data by humans is expensive and needs expert knowledge.

Reinforcement Machine Learning: In this learning scenario, the interaction with the environment plays a crucial role to build the model [5]. Reinforcement machine learning scenario aims at maximizing the rewards by involving an online performance evaluation in the learning process. In such a learning process, the model will react to the evaluation feedbacks aiming at increasing the rewards and achieving the best performance. Reinforcement machine learning has become more important in recent years, as it produces the best solutions in a lot of world wide used applications, for instance helicopter flying [176], resource-constrained scheduling [49], robot control systems [59] and playing backgammon [111].

2.1.2. Performance Evaluation

While performing machine learning tasks, the predictive performance of the model has to be somehow measured. This is done by defining an adequate error function for the given problem setting. Because this thesis discusses the forecasting of time series datasets an

adequate error function for the problem "forecasting of time series data sets" has to be defined.

In this section, the commonly used forecast error measurements are introduced and discussed. When comparing different forecasting models for forecasting a time series dataset, the predictive performance of those models needs to be compared and the lowest forecast error is always preferred. To achieve such a comparison, an error metric is required. These metrics measure the error in the model by comparing the actual values with the predicted values. The difference between different metrics lies in the way of computing those errors [32][72]. In the following, $\hat{x}$ represents the predicted value, X represents the real value, and N is the number of samples in testing set.

Mean Absolute Error (MAE): The MAE as defined by Equation (2.1) is normally used to identify the average error. This value is more likely used as metric when the errors are uniformly distributed over all forecasted values.

$$MAE = \frac{1}{N}\sum_{h=1}^{N}|X_h - \hat{x}_h| \tag{2.1}$$

Root mean square error (RMSE): The RMSE as defined by Equation (2.2) is used to evaluate in average the performance of a certain model [32]. The sensibility to outliers in the data is considered as a concern when using RMSE. Moreover, it gives more weight to large error since it squares it. However, it has an advantage over the MAE metric because it doesn't compute the absolute error and is very suitable when the errors have normal distribution.

$$RMSE = \sqrt{MSE} \tag{2.2}$$

$$MSE = \frac{\sum_{j=1}^{J}(X_j - \hat{X}_j)^2}{J} \tag{2.3}$$

where: X_j is an observation and $\hat{X}_j$ is the corresponding predicted value.

According to Equation (2.3), the MSE is calculated as the sum of the squared forecasting errors divided by the number of the observed values.

Mean Percentage Error (MPE): The MPE as defined by Equation (2.4) gives us the ability to decide whether the predictive model estimates values lower than the true values or higher them. If the sign of MPE is negative, then the model predicts values higher than the true values i.e., it "over-predicts". Nevertheless, if the sign is positive, then the model estimates values below the true values i.e., it "under-predicts".

$$MPE = \frac{100}{N}\sum_{h=1}^{N}\frac{X_h - \hat{x}_h}{X_h} \tag{2.4}$$

Mean Absolute Percentage Error (MAPE): The MAPE as defined by Equation (2.5) evaluates the performance of a forecasting model by calculating the mean absolute percentage error. This makes it more interpretable to any one even without a previous knowledge about the modeling problem.

$$MAPE = \frac{100}{N} \sum_{h=1}^{N} \frac{|X_h - \hat{x}_h|}{X_h} \tag{2.5}$$

MAPE as well as MPE are not suitable when there are zero values or even very small values in the ground truth since they are calculated by dividing the difference between predicted value and actual value over the actual value. Hence, large values and undefined values in certain situations, namely by dividing over zero can be obtained. Therefore, we use Mean Error Relative (MER) as an alternative metric.

2.2. Big Data Software Environments

Driven by the rapid growth of Big Data in scientific and industrial domains, efficient handling of large amounts of data has become an important research topic [133][108][35]. Big Data is often characterized by three properties, namely volume, variety and velocity as seen in Figure 2.2. Handling data showing all these properties together will make the process of managing, revealing and gaining insight, knowledge and information from the data more complex and a challenging task. Typically, larger scale computing clusters are needed for handling Big Data, because the data couldn't be stored on a single storage system or be analyzed on a single computer.

The large amount of high dimensional data leads to the problems of high computational costs and instability in the performance of the learning algorithms. Moreover, gathering such data from multiple data sources using different techniques leads to the challenges of heterogeneity in data. To overcome these challenges, the need for software solutions for data processing and analysis which make optimal use of high speed networking, large capacity storage and scalable distributed/parallel processing increased dramatically. Such software solutions and frameworks are typically called Big Data frameworks.

One of the best known open-source frameworks is Apache Hadoop which supports big data processing and storage in a distributed computing environment [149]. It encompasses various components including a distributed file system, the data processing tool MapReduce and a cluster resource manager. Besides enabling the reliable storage of extensive files in a cluster, the Hadoop Distributed File System (HDFS) provides fault tolerance by splitting the files into blocks and replicating these blocks multiple times over the cluster. It provides fault tolerance by splitting the files into blocks and replicating these blocks multiple times over the cluster.

Figure 2.3 shows the architecture of HDFS which consists of a Name Node and multiple Data Nodes. The name node coordinates the operations of the underlying file system (e.g.

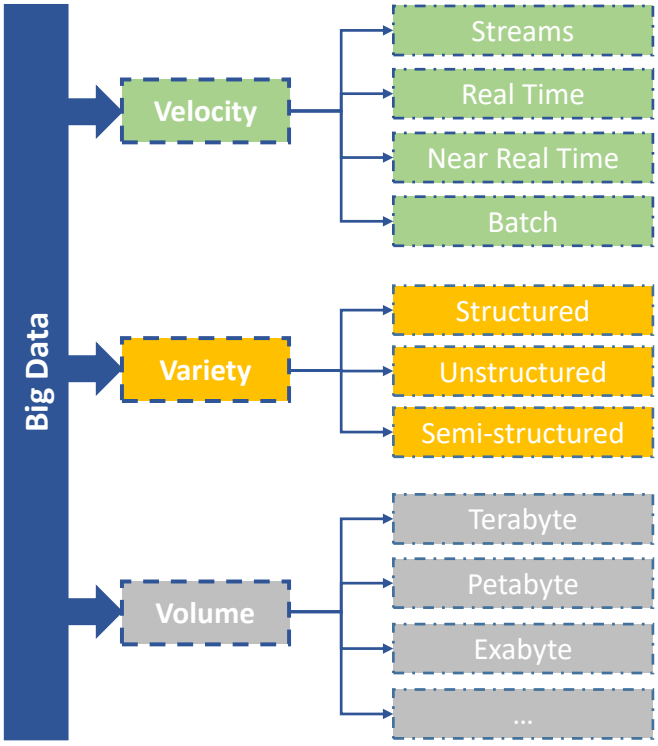


Figure 2.2.: Characteristics of Big Data.

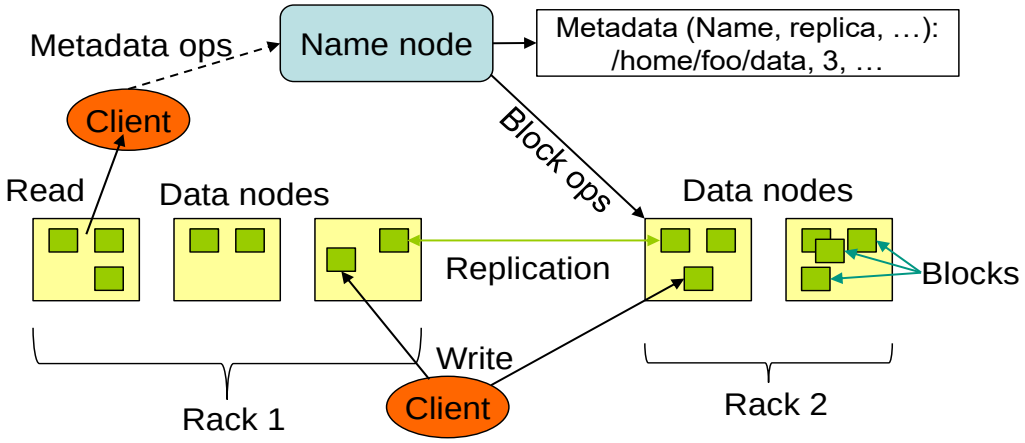


Figure 2.3.: HDFS architecture [24].

opening and closing files, etc.) and manages metadata maps which provide the knowledge on which piece of a file can be found on which Data Nodes. On the other hand, the data nodes store the file blocks and serve the read as well write requests. For reliability, also the Name Node functionality can be set up redundantly.

Hadoop MapReduce [40] is one Big Data processing framework which allows parallel processing of data according to the well known Map-Reduce data processing method. This method processes data using two separate steps, i.e. a first “Map” step responsible for

transforming data into key/value pairs and then a second step Reduce, which accepts the output from the Map task as input, and aggregates the data from the map somehow to produce the final output of the Map-Reduce operations.

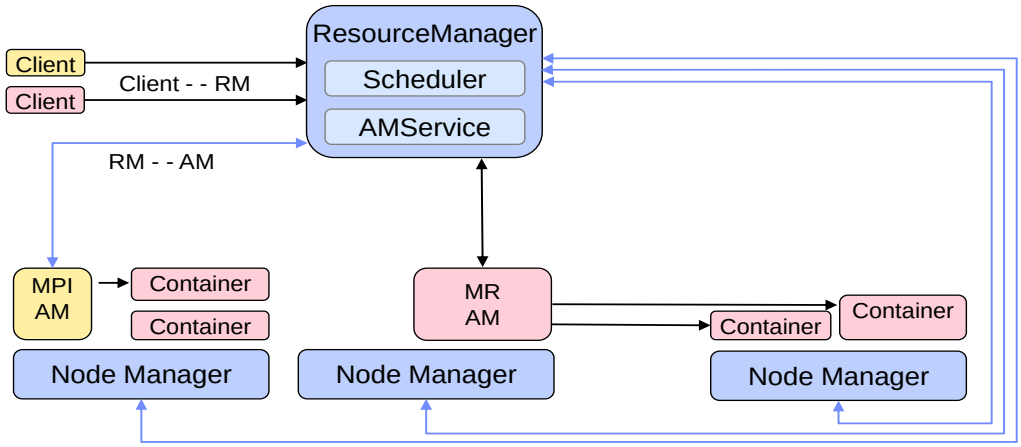


Figure 2.4.: YARN architecture [80].

One of the state-of-the-art technologies in resource management is Yet Another Resource Negotiator (YARN) [167]. It is based on the idea of decoupling the application and the required computational resources e.g. CPUs, RAM, etc. Figure 2.4 illustrates YARN's architecture which is mainly composed of a Resource Manager (RM), multiple Node Managers (NM) and an Application Master (AM) for each program. When an application is submitted to the RM, the RM allocates a container accommodating the required resources for the application and contacts the related NM to launch this container. The container then executes the Application Master (AM) which coordinates the application scheduling and task execution and sends resource requests to the RM.

Another Big Data Analytics framework is Apache Spark [134], which also supports the Map-Reduce processing method but in general provides a more generic and more performant parallel processing framework which also comes with a library of ready to use ML models already implemented. The Map-Reduce implementation container in Spark is several factors quicker than the original implementation in the MapReduce framework and also more reliable.

2.3. Microservices

Developing web applications using a monolithic architecture where the database, the server and the client code are maintained in a single codebase have drawbacks if the application gets more complex in terms of maintainability and their ability for adding new functionality quickly and easily. If the application can only be deployed as a whole, it is also limited in terms of scalability. With the upcoming of cloud technologies, more and more companies like Amazon, Netflix and Zalando have shifted from a monolithic architecture

to a newer more scalable and maintenance friendly architecture called microservices architecture [2][18][39]. In this section, the basic background required to understand the advantages of a microservice architecture is provided. For the sake of simplicity, the term service in this thesis refers to microservice.

2.3.1. Characteristics

The main idea behind microservices is that the whole application is decomposed into multiple smaller services or runtime artefacts where each of them can be deployed and run as an operating system process independently from the others. The decomposition of the functionalities into services (runtime artefacts) are done according to the application's business functionalities. Each service follows the Single Responsibility Principle (SRP) and implements only a certain business functionality [88]. By following the SRP, the services become highly cohesive and decoupled, leading to an easier code maintainability. In contrast to that, the monolithic applications lack hard boundaries and, with added functionality, tend to become complex and tightly coupled which in turns leads to difficulties when changes are made since they often span multiple components.

Another advantage of microservices is that they do not require the redeployment of the whole application when implementing new features or fixing bugs. Instead, only parts of the application including the corresponding service need to be adapted and redeployed. Furthermore, microservices of a single application are not constrained to be implemented with the same set of technologies and frameworks. This allows teams working on different microservices to use independent technology stacks resulting in different programming languages and data storage technologies suitable to the data they process.

2.3.2. Bounded Contexts

It is of great importance in the design and implementation process of a microservice-based application to identify the scope of each microservice in the application. To find the cohesive and loosely coupled boundaries in a system, a pattern called bounded context which originated in Domain-Driven Design (DDD) is often utilized as a guideline [95]. Before describing the bounded context, the terms domain, domain model and subdomain which are integral parts of DDD, first need to be introduced. The domain can be defined as a sphere of knowledge, influence or activity [88]. Essentially, the domain is the problem space that the system addresses. The domain model is used to depict the key elements of a domain. This is done by establishing a ubiquitous language, an important communication tool between developers and domain experts containing a fundamental knowledge about the domain.

To facilitate the modeling of complex applications, the domain is usually decomposed into subdomains. Each one of them is responsible for a separate business capability. These subdomains are then mapped, preferably one-to-one, to bounded contexts which describe the solution space of the system [120]. The main idea behind defining a bounded context is

to form the explicit boundaries of a domain model, delimiting its applicability to a specific context which helps team members to have a shared understanding of what needs to be consistent and how it relates to other contexts.

Although the bounded context is a favored approach to design a microservices architecture, the scopes of the services are often affected by the boundaries of an organization. This effect is called Conway's law which states that organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations".

2.3.3. Communication Types

Driven by the nature of a microservices architecture in which the services are isolated from each other and distributed over a network, the communication with and between microservices are becoming the most relevant gluing part to assemble the set of microservices into a full application. Therefore, it is often said that microservices should have smart endpoints and dumb pipes, meaning that the smart logic should be inside of the services and only lightweight mechanisms and standards should be used for their communication [41].

Communication styles are usually divided into request/response and event-based techniques [95]:

- Request/response: in this technique, two services can directly communicate with each other, where one service initiates a request to another and in return expects a response.
- Message-based: Using message-based communication, a communication partner creates a message which is then sent to a Message Oriented Middleware (MOM) which forwards the message asynchronously to one or more receivers. Receivers typically register themself in a MOM for receiving messages from certain types of message queues.
- Event-based: this is a special form of message-based communication where the sent message corresponds to events, where one service or producer communicates an event as a message to a MOM and all services that have subscribed to the event type as message type will receive the message.

Typically request/response based communication will be instrumented by a microservice application for directly invoking a certain business functionality provided by a service, e.g. a user interface uses the business capability of a certain background service. It can also be used when one service has to use the business functionality provided by another service. For implementing such request/response interfaces, nowadays often REST-based communication interfaces are used which are very "lightweight". Asynchronous event-based communication is often used in a microservice architecture to synchronize state or more generic data between several microservices. The event-based MOM framework therefore

often also implements coordination algorithms for performing necessary coordination between the distributed instances of the microservices.

2.3.4. REpresentational State Transfer (REST)

To implement the request/response communication style, microservices follow the REST (REpresentational State Transfer) principles, a protocol-agnostic architectural style that commonly uses HTTP as a communication protocol. The term REST was first coined by Fielding et al. in [46] and is made up of the following 6 constraints.

1. Client-server: to improve the portability of the client i.e. user interface and scalability of the server entities, the client and server should be separated. This constraint enables the independent involvement of both.
2. Stateless: this constraint affects the communication between the client and server and declares that it should be stateless, meaning that the client requests to the server must contain all necessary information.
3. Cache: improving the network efficiency by requiring data within a response to be labeled as cacheable or non-cacheable.
4. Uniform interface: this constraint emphasizes the importance of a uniform interface between components. To obtain it, a variety of interface constraints are defined e.g. identification of resources, manipulation of resources through representations, self-descriptive messages and hypermedia as the engine of application state.
5. Layered system: to reduce the complexity of an overall system, hierarchical layers should be implemented which constrict the components' behavior.
6. Code-on-demand: this is an optional constraint that allows client functionality to be extended by downloading and executing code in form of applets or scripts.

As already said, REST-based communication interfaces (also called REST-APIs), will be typically implemented using web technology and therefore HTTP(s) as request/response protocol. URL patterns together with HTTP header fields and optionally a HTTP payload define a request to a web server hosting the service which typically returns a HTTP response consisting of a status code, HTTP header fields and a payload in an adequate Multipurpose Internet Mail Extensions (MIME) format [136]. The payload of a response or a request has often a machine and human readable structure which is often technically implemented by using an application specific JSON or XML format. The usage of such a communication interface requires only a small framework for performing HTTP calls and creating, interpreting and manipulating JSON and/or XML, and is therefore very lightweight. Such REST frameworks exist for all common generic computer languages and REST API's are typically designed so that they are completely independent from the underlying operating system and hardware environment. This makes REST services accessible from any platform with any language and this makes REST services ubiquitous.

2.4. Time Series Datasets

A time series is defined as a sequence of values (univariate) or data tuples (multivariate) which is indexed and ordered by points in time (often described by a timestamp), which typically correspond to observations (measurements) made at that point in time. Time series are used for a wide range of application scenarios, such as energy, weather and finance. In this thesis different time series datasets are used to evaluate the proposed meta learning concepts for the ML use case "forecasting of time series" and obtain the desired results. Therefore, several sets of time series data will be used which will be described in the following subsection.

2.4.1. ENerGO+

The ENerGO+ software system was set up at the KIT Campus North, the former Karlsruhe Research Center and now a Helmholtz research center, as central measuring system which records the consumption of Energy by the KIT institutes at Campus North. The datasets collected in this system contain time series for the energy consumption of building parts or technical plants for electricity, gas, thermal energy from the district heating system and water.

All time series consumption data gathered between 01- 01-2006 and 06-08-2018 available through web-based user interface ENerGO+ were extracted. About 70 GB of raw data used in this thesis for evaluation was provided in CSV format which contains a variety of different measurements. For providing a interpretation context for each dataset, a metadata file was provided with additional information about each measuring station and the type of consumption data contained in the time series:

- Stdid: unique identifier of measurement time series.
- Anlage: identifier of the building the measuring station is located in.
- Teilanlage: sub-division of the building or a single technical plant.
- Medium: Indicates what is measured. For example electricity, gas or water.
- Erfassungsart: Type of acquisition: manual or auto.
- Bm_zw_ausgang einheit: Unit of measured medium. For example kWh for electricity or m3 for water flow

For the evaluations in this thesis, only electricity consumption is of interest, so all other measurements were discarded. Because equidistant measurements are easier to handle for forecasting, only electricity meters with equidistant auto-acquisition of data were considered. Each electricity meter at the Campus is configured to provide energy consumption data every 15 minutes. From the 2125 time series all but 761 were discarded in this pre-selection phase. To further exclude unusable time series, a deeper analysis of the actual data was required. We decided to not use time series data with a time span of less

than one year, which also lead to a few discarded short time series. Some time series also contained large gaps (maybe, because some meters were defect over a longer period in time) which is also problematic. This led to the decision to discard all time series with gaps greater than two weeks. Additionally, we decided to only use time series with less than 2% of the data missing, therefore a few more time series had to be discarded as the pre-processing of the data progressed. Finally, to enhance the quality of the data even further - depending on the evaluation use case - several automatic quality enhancement procedures e.g. interpolation missing data points were performed prior to the evaluation. 200 time series datasets are resulting from these filtering steps. Further details about these will be provided later in this work in the corresponding evaluation chapters.

2.4.2. Ausgrid Solar Home Electricity Data

A further dataset used for some evaluations was the Ausgrid Solar Home Electricity Data Set. This time series dataset provided by the state-owned Australian energy provider Ausgrid [1]. It offers time series observations for 300 solar customers with installed PV systems. All time series extracted from this dataset with time samples n= 1, ..., N have a temporal resolution of 30 minutes and contain measurements from 01-06-2010 to 30-07-2013.

For each customer three readings categories were recorded at half-hourly intervals:

- GC: General Consumption.
- CL: Controlled Load Consumption.
- GG: Gross Generation.

Additionally, for every customer the generator capacity of the installed solar systems was recorded in Kilowatt Peak (kWp). In this work we used only GG.

2.4.3. Weather Time Series Dataset

For the evaluation scenarios using the generation data from the Ausgrid Solar Home Electricity dataset, also corresponding weather data was important. A corresponding time series weather dataset is provided by the Bureau of Meteorology of the Australian Government [2]. The measurements were carried out at a weather station in the city of Sydney, which lies in the same area where the houses for capturing the time series dataset explained in 2.4.2 were located. The weather dataset contains observations for the same time frame as the power generation time series dataset from section 2.4.2. Namely, day-to-day observations for evapotranspiration, maximum and minimum relative humidity, maximum and minimum temperature, precipitation and solar radiation are provided.

[1] https://www.ausgrid.com.au/
[2] http://www.bom.gov.au/

3. Related Work

A variety of learning algorithms, methods and approaches are already offered for applying ML models to different use cases whereby the selection and adaptation of a good performing model is still a complex, long lasting and error prone process [166]. To simplify the usage of ML, the ML community therefore also developed powerful techniques, frameworks and tools to make the usage of ML more accessible to end users. In this chapter, several research projects with the main goal of supporting users in performing ML tasks in Big Data environments are presented. Section 3.1 summarizes a group of research projects for facilitating and managing ML tasks. These projects are mainly categorized into data analysis and ML workflow management frameworks. Section 3.2 covers another group of research projects using meta learning to help the user in finding an adequate model for a given ML task. This section introduces projects focusing on the type of meta features, algorithms instrumented as meta learners and methodologies used to achieve the main goal of automated model selection. In Section 3.3, state-of-the-art research projects existing in the literature for the purpose of generating new time series datasets to augment the set of training data are introduced.

3.1. Machine learning software and tools

The process of selecting, configuring and training a good performing ML model for a given use case is still characterized by an iterative trial-and- error procedure where in each iteration, the ML user discovers new essential insights into the effectiveness of certain configuration settings on the model and thereby influencing future experiments. To aid users to go in the right direction, many frameworks are developed which help users by providing ML frameworks with already given configurable model implementations, or tools for organizing and analyzing this trial-and-error approach. Such frameworks can be categorized into data analytic and ML workflow management as well as visualization frameworks.

3.1.1. Data analytic framework

Frameworks like Apache Spark [187] which is a data analytic framework containing a good library for more traditional ML algorithms, or TensorFlow [110] which is dedicated to deep learning, are low level frameworks that help data scientists in programming ML algorithms which could then be executed on a local computer or even for better performance on a

computing cluster. Such frameworks typically don't provide easy-to-use user interfaces for non-experts by themselves but there are additional (Open Source) tools (e.g. Jupiter Notebook [73]) which provide lean web user interfaces to such frameworks for hiding the details of the background cluster runtime environment from the user. Typically, these interfaces are aimed towards more experienced data scientists and less towards non-expert users who just want to apply ML algorithms.

Contrary to the tools aimed for the experienced ML programmers, there are nice User Interface (UI)-based tools targeted to non-expert users with little to no programming experience at all. Johanson et al. in [60] developed OceanTEA, a framework to analyze time series datasets in a climate context. OceanTEA leverages web technology such as microservices and a nice web UI to interactively visualize and analyze time series datasets. It is a cloud-based software platform, consisting of a microservice back-end and a web UI, similar to the framework implemented in this thesis. Both components communicate with each other through an API gateway utilizing REST communication and each microservice is deployed independently using container automation through a Docker. OceanTEA provides four main UI interfaces for the exploration and analysis of oceanographic times series data including functionalities of time series data management, data exploration, spatial analysis and temporal pattern discovery.

Another project focused on the acceleration of research in energy data management and analysis is WattDepot presented by Brewer and Johnson in [26]. This software platform is also open source and internet-based. It supports the collection, storage, analysis and visualization of data coming from energy meters. The architecture encompasses three types of services, namely sensors, servers and clients. The sensors collect the data from different energy meters and send it to the services which store the incoming data by utilizing the provided RESTful APIs. Since the services are not coupled to a specific database, flexible data storage options are provided. For analysis and visualization, the clients request the data from the services in the format XML, JSON or CSV. The applications of WattDepot include a web application for a dorm energy competition and a power grid simulation mechanism.

However, both WattDepot and OceanTEA typically provide hard-coded dedicated ML based analysis features which are specifically tuned towards specific application use cases and therefore e.g. other ML tasks such as forecasting which is needed in the energy application field are not included in them.

Shrestha et al. in [148] developed a user-friendly web application to analyze health and education datasets. This tool also includes ML algorithms for the forecasting of time series data. The application also has a nice and easy-to-use user interface that was developed using human-computer interaction design guidelines and principles and targeted at novice and intermediate users. The technologies used were Java, the Play framework and Bootstrap. But only linear regression, logistic regression and back propagation were utilized to perform forecasting on the input datasets. However, this framework is also not able to solve more complex ML tasks by using Big Data analysis frameworks executable on cluster computing environments, and it can only be used as standalone application on a desktop computer.

Apache PredictionIO [33] is an open source ML framework for developers. Besides supporting the deployment of ML algorithms, Apache PredictionIO allows expert users to train and test ML models and query results via RESTful APIs. It is built on top of state-of-the-art scalable open source Big Data frameworks, e.g. Hadoop, HBase, Elasticsearch and Apache Spark. The drawback here is the non-existence of an easy to use UI layer to facilitate performing ML tasks by non-programmers.

With the increasing interest in ML, a new market called ML as a Service (MLaaS) is existing, whereby ML functionalities like training and deploying ML models are typically provided through web services. Ribeiro et al. [124] introduced a scalable and flexible open-source MLaaS architecture, including a graphical user interface (GUI), for building multiple predictive models from various data sources simultaneously. It is based on a service component architecture and is implemented using Node.js and JSON. The architecture was evaluated by implementing three algorithms; Multi-Layer Perceptron (MLP), Support Vector Regression (SVR), and K-Nearest Neighbors (KNN). The resulting models were compared based on their accuracy using mean absolute errors and mean squared error as well as their computing performance, which the GUI visualizes in addition to their predictions. Another leading cloud-based MLaaS platforms is google cloud machine learning engine. This platform simplify the usage of ML by abstracting away the many challenges related to ML including making the infrastructure more affordable and scalable [182]. The distinguishing factor compared to the framework developed in this thesis is that these MLaaS platforms at least until now do not provide an integrated model management, and therefore the possibility to store and retrieve ML models for future usage.

3.1.2. ML workflow management and visualization frameworks

Model and data versioning is also an important area of research that has produced systems to manage the process of building ML models. The process of building a satisfactory ML model by a data scientist is characterized as an iterative trial-and-error procedure, where in each iteration the user reveals essential insights into the effectiveness of algorithm configurations. Since the models may become numerous, it is important to keep track of the relevant information so that the model's performance with different configurations can easily be tracked and analyzed. This leads to the need of an efficient model management which encompasses the storage and retrieval of the models and related metadata (e.g. hyperparameters, evaluation performance, etc.) in order to analyze them collectively [166].

Multiple recent research projects have been introduced addressing model management as a part of the ML workflow. Vartak et al. in [166] introduced ModelDB, a system for tracking and versioning ML models in the form of pipelines. The authors argued that data scientists are reluctant in using other environments than their favored ones, especially those with a GUI and therefore they provide native client libraries for scikit-learn and Spark MLlib which can be used to track and store models and related metadata. The framework consists of a front-end and a back-end encompassing a relational database and custom storage engine. The front-end is implemented as a web UI and supports the review,

inspection and comparison of the tracked and indexed models and pipelines through a tableau-based interface. In addition, the information can be explored and analyzed using SQL. The limitations here are that ModelDB is developed as a monolithic application making it difficult to be maintained and further developed. Moreover, ModelDB do not provide the ability to handle problems in the context of Big Data.

Vartak et al. [165] introduced another system to analyze ML models built with scikit-learn and Tensorflow called Mistique (Model Intermediate STore and QUery Engine). It captures, stores, and provides the ability to query model intermediates like the input data. Mistique is implemented in Python and consists of three main elements, being a PipelineExecutor, a DataStore, and a ChunkReader. It utilizes a column-based schema for its DataStore which consists of an in-memory store and a persistent store. Also, a so-called MetadataDB exists which is a central repository for the metadata of the pipelines and intermediates. The authors focused on efficient storage and proposed two optimization strategies for ML pipelines. First, columns that are found to be similar or identical are compressed. Secondly, a query cost model and a storage model were implemented to determine if a model should be rerun or the intermediate read and if an intermediate should be stored. Finally, the authors assessed the storage gains and speedup for ML pipelines and deep neural networks. Furthermore, the cost models were evaluated and found to be effective. Also the overhead of utilizing the framework was estimated by comparing the runtime performance of different pipelines.

Schelter et al. [137] introduced a system for auditing the ML workflows of more general model types (e.g. neural networks), including the support for dataset schema management. The system consists of a back-end running serverless on AWS and offers REST APIs for communication, but unlike [165] it does not include a web UI. The metadata is stored in a document database and the system is integrated with scikit-learn, Spark MLlib and MXNet. For decoupling purposes, the authors chose a declarative approach where artefacts of the workflow are described by metadata and not via compiled code. Additionally, they guaranteed consistency by applying the immutability principle meaning that items are only recorded once.

To manage ML models and their lifecycle, MLflow is introduced in [185]. Expert users can develop and track ML experiments as well as share and deploy ML models. MLflow is developed as an open source software system addressing typical problems of ML workflow management, particularly experimentation, reproducibility and deployment. MLflow supports programming of models with Python, Java and R, and provides REST APIs encompassing three main elements. The first one, MLflow Tracking, offers APIs for logging experiments and supports querying the results through APIs as well as visualizing them with a web UI. The second component, MLfow Projects, can be used to create reusable software environments for reproducibility and is configured through YAML files. The last item, MLfow Models, provides the functionality to package ML models in a generic format and deploy them. Those models incorporate similarly to MLflow Projects a YAML file which contains the metadata of the model.

To address the issue of model deployment, a variety of frameworks and tools are developed. Tensorflow serving [106] provides a flexible and powerful system for serving tensorflow

models on google's cloud platform. It allows expert users to achieve an efficient integration of tensorflow models in production environments. Kubeflow [23] is a cloud platform for ML built on top of google's internal ML pipelines. It provides expert users with a lot of functionalities including notebooks for training and serving tensorflow models. H2O Flow [27] is another efficient framework for creating and managing ML and deep learning workflows including training and testing models. This framework supports Python, R and scala on top of Hadoop/Yarn and Apache Spark.

3.2. Meta learning for energy time series model selection

To select ML models following the well-known trial-and-error approach, the relevant configurations of different learning algorithms are changed and tested until a model with good performance is found. Consequently and due to the large number of available ML algorithms and their relevant hyperparameters, this process is becoming more complex and even difficult for non-expert users. To tackle this challenge, meta learning approaches have been proposed [164][70][163][171]. The main aim of meta learning is to find indicators that map datasets to the best suitable algorithm for performing a certain task (e.g. forecasting). To this end, meta learning uses a set of attributes, referred to as meta features, to capture the characteristics of the data mining task and searches for the correlation between these features and the best machine learning algorithm for performing a given task.

While several studies have investigated the use of meta learning to select the most appropriate model, the majority of them studied the selection of classification algorithms, e.g. [70][122][127][128][45][175][132][74][191][85][123], to name a few. With the growing popularity of regression, the first use of meta learning in the context of time series was by Ludmir et. al. in [118] who proposed an approach for time series model selection. Two case studies have been investigated in their work. In the first one, the authors used a single machine learning algorithm to select models for forecasting stationary time series. In the second one, the well-known NOEMON approach [63] to select time series models for the M3-competition has been used. As meta features, a set of 10 meta features including simple, statistics and time series meta features are extracted to describe time series datasets.

In [98], Wang et. al. proposed a meta learning framework for recommending the most appropriate forecasting method from 4 different candidates, namely random walk, exponential smoothing, neural networks and ARIMA. As meta features used to characterize time series datasets, serial correlation, kurtosis, strength of trend, nonlinearity, strength of seasonality, skewness, periodicity, self-similarity and chaos are extracted. A decision tree algorithm has been used as a meta learner to recommend one of the aforementioned candidates for an input time series dataset based on its characteristics.

The same group of meta features has been later used by Widodo in [172]. The difference is that the author tried to reduce the time series dimensionality by applying Principal Component Analysis (PCA). Kück et. al. in [76] used feedforward neural networks as a meta learner to select the best time series forecasting model for 78 time series from the

NN3 competition. As algorithm candidates, single, seasonal, seasonal-trend and trend exponential smoothing were used. To characterize time series datasets, error-based features (landmarkers) and statistical tests were used as time series meta features.

To achieve the main goal of meta learning in finding the mapping between meta features and the learning algorithm, different algorithms as well as approaches have been proposed as meta learners in literature. On the one hand, some of them used statistical methods to induce meta learners [138][93]. On the other hand, neural networks, decision trees and other computational intelligence methods [118][98][168] are used as meta learners. In the cited scientific works, different categories of meta features such as simple, statistical, time series, model-based and landmarking are utilized. A good overview summarizing the state-of-the-art of meta learning approaches applied for time series forecasting can be found in [151]. However, it is noticeable that although a large number of scientific papers address the topic of meta learning, only a few are dedicated to take the advantage of it in the case of energy forecasting.

Building Energy Model Recommendation System (BEMR), a meta learning based framework to recommend the most appropriate forecasting algorithm for building energy profiles based on building characteristics, is proposed by Cui et.al. [38]. As meta features, physical features of the building combined with statistical and time series meta features extracted from the operational and energy consumption data of the building were used for constructing efficient meta examples to be used by the meta learner. While the algorithm selection is fully automatic, this approach has the disadvantage that the meta feature set uses some physical construction properties that are often not available in praxis due to privacy issues.

In the following, some meta learning approaches developed as frameworks with wizard are discussed. A parallelized, component-based, modular and easily extendable meta learning system for univariate and multivariate time series load forecasting is described in [91]. Here, Matijaš et. al. built an ensemble of euclidean distance, CART decision tree, LVQ network, MLP, AutoMLP, e-SVM and Gaussian Process (GP) algorithms to find the association between the meta features and the forecasting performance. Minimum, maximum, Standard Deviation (SD), skewness, length, periodicity, highest ACF, traversity, kurtosis, granularity, exogenous, periodicity, trend and fickleness are considered as meta features. These features were weighted with the Relief feature ranking method [153] before being utilized by the ensemble-based meta learner.

Auto-WEKA [158] is a framework for automatically selecting classifiers and hyperparameters implemented in WEKA. In the updated version Auto-WEKA 2.0 [75], the selection of the best regression algorithm is provided. To solve the Algorithm selection and hyperparameter optimization (CASH), bayesian optimization is utilized. Due to the large number of hyperparameters that can be tested while building ML models, such hyperparameters are structured as trees or as Directed Acyclic Graph (DAG). In the evaluation, two Sequential Model-Based Optimization (SMBO) algorithms, namely Sequential Model Based Algorithm Configuration (SMAC) and Tree-structured Parzen Estimator (TPE) are used. As a baseline for the evaluation, the authors also used two algorithms which do not perform hyperparameter optimization but only algorithm selection: exhaustive evaluation

and Hoeffding race. WEKA provides 47 classification algorithms grouped into 30 base classifiers, 14 meta methods and 3 ensemble classifiers. The authors used 10 benchmark datasets obtained from the UCI repository for evaluation. Each dataset was partitioned into a 70/30 train/test random split and the hyper-parameter configurations were evaluated based on standard 10-fold cross validation. As a result, the SMAC algorithm as a Bayesian optimization method performed best.

The same principles of Auto-WEKA are used in Auto-Sklearn [45] which is a a meta learning framework based on scikitlearn. To solve the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem, they built on the research from Auto-WEKA and used the same Sequential Model based Algorithm Configuration (SMAC) algorithm as Bayesian optimizer for hyperparameter tuning. To further improve the results and increase the accuracy and robustness of the framework, Feurer et. al proposed a meta learning approach to suggest some models which in turn are used as seed for the optimization process. In an offline phase, they collected meta features and performance data of 140 datasets from the OpenML repository, which is used to find the candidate models for new datasets. The implemented 38 meta features include simple, information theoretic and statistical features, but do not include landmarking meta features as they are too computationally expensive to calculate. Performance evaluation was done using the available 15 classification algorithms from scikit-learn.

As a benchmark evaluation, Auto-Sklearn is compared to Auto-WEKA which resulted in a significantly better performance in some cases and a comparable performance in most of the other cases. They also performed different experiments to evaluate the meta learning and ensemble-generating approach which showed that both methods lead to a better performance than vanilla Auto-Sklearn. The drawback in Auto-WEKA and Auto-Sklearn is that they are implemented as monolithic applications which limit the scalability and increase the difficulty of maintenance. Moreover, they did not provide the possibility to handle model selection for large amounts of data.

Another meta learning framework developed with the R language is SmartML [87]. It is implemented as a web application with REST APIs. SmartML can recommend a classification algorithm including hyperparameter tuning based on a total of 25 meta features. The limitation here is also that SmartML does not support the usage of Big Data ML frameworks for large scale processing on computing clusters. In contrast to the aforementioned works, the meta learning framework presented in Chapter 7 is developed as a microservice architecture runnable on a cluster computing platform to increase the scalability and facilitate maintainability. Moreover, it utilizes powerful Big Data ML frameworks to perform model selection in cluster computing environments.

To the best of our knowledge, all meta features used in meta learning for energy can be subcategorized into simple, statistical, time series and domain-based meta features. Simple meta features can be directly derived from time series datasets, e.g. the number of samples and the number of attributes. Statistical meta features capture the statistical properties of time series datasets such as cor and skewness, to name a few. Time series meta features introduce in-depth insight into the characteristics of time series datasets, such as acf_-features, pacf_features, arch_stat, crossing_points, heterogeneity, hurst, max_kl_shift as

explained in [82]. Domain-based meta features, as its name implies, are derived from the domain, in which the time series readings are collected, for example, the physical characteristics of the buildings [38].

3.3. Generating new time series datasets

In this section, we cover some of the related work for generating new artificial time series datasets. Talagala et al. [156] used a model-based time series generation technique to increase the size of the original time series training dataset. Autoregressive integrated moving average (ARIMA) and Exponential Smoothing (ETS) models were trained for every time series and then a number of new time series were generated with each of these models via simulation. To visualize the characteristics of the time series and compare the distribution of the generated time series with the distribution of the original ones in the meta feature space, Principal Components Analysis (PCA) is used. As a result, they found that the new simulated time series increased the diversity and evenness of the dataset in the meta feature space.

Another method for generating synthetic time series was proposed in [47]. This method is based on differently averaging a set of time series from the same predefined class. The averaged time series are then treated as new synthetic time series. The weights of the time series for the average can be varied, making it possible to create many new time series datasets. For calculating the average of a time series dataset, three different methods based on DTW Barycentric Averaging (DBA) are proposed. The three methods are called average all, average selected and average selected with distance. Each one of them distributes the weights differently. Using the average selected with distance method, they doubled the number of time series in each class for every dataset in the University of California Riverside (UCR) archive. In an evaluation study on 85 datasets, they showed that the augmentation increased the accuracy of a 1-NN DTW classifier on 56 datasets on average of 3.81%, decreased the accuracy on 22 datasets on average of -1.72% and didn't change the accuracy on 7 datasets. Using the Wilcoxon signed rank test, they found that the increase in accuracy was statistically significant at the 0.001 level.

A genetic algorithm for generating new time series based on user-defined meta features is proposed by Reif et. al in [121]. The generation process involves the minimization of the cost function $f(x)$ where x is the meta feature vector of the current time series. The genetic algorithm mutates the time series by shifting data points and recombining them by swapping fractions of them. Note that this algorithm gives the user more control over meta feature combinations than the algorithm described in [65], where only time series with meta feature vectors on a 2-dimensional Principal Component (PC) space can be targeted.

To learn the probability distribution of a real smart grid dataset and then to generate synthetic time series, a Generative Adversarial Network (GAN) is used in [188]. The goal is to make the newly generated time series datasets statistically in-differentiable from the

original ones. To train GAN, the authors used energy consumption and solar generation records from 93 different users. To assess the similarity between the newly generated time series datasets and the original time series, Maximum Mean Discrepancy and Dynamic Time Warping (DTW) based k-means clustering as well as Short-term Load Forecasting (STLF) forecasting errors of ARIMA models are used. The proposed approach was able to learn the conditional probability distribution of input features and new samples are produced based on the learned distribution.

In [64], Kang et al. proposed GeneRAting Time Series with Diverse and Controllable Characteristics (GRATIS), a new approach to generate time series with diverse characteristics by using Gaussian Mixture Autoregressive models (MAR). This approach is able to simulate multi-seasonal time series. They showed this by simulating the half-hourly electricity demand based on an original time series. In contrast to [65], they used a genetic algorithm to tune the parameters of a MAR model so that the time series which are generated by the MAR model gets closer to the target meta feature vector, instead of changing the time series itself in a space whose dimensionality is equal to the length of the time series.

3.3.1. Summary

In this chapter, we gave a detailed overview of the state-of-the-art research projects related to our work. We started in Section 3.1 by briefly presenting a group of ML software tools aimed at facilitating and managing performing ML tasks. Many ML libraries (e.g. scikit-learn [113], R [157], Spark MLlib [94]) provide the ability of training ML models, but they require the user to have an understanding about the algorithm and are targeted to expert users. However, making ML more accessible to a broader group has become an active research topic, which can be explained by the increased interest in ML by non-than expert users. Table 3.1 summarizes the aforementioned state-of-the-art solutions presented in Section 3.1 highlighting their properties and differentiating them from the solution presented in this thesis.

As seen in this table, none of the frameworks and tools presented in Section 3.1 perform automated model selection of ML models. Some of them support non-expert users (category 2A) presented in Table 1.1, for example OceanTea, WattDepot and ML Flow. However, the other category of non-expert users, namely 2B, still represents a challenge that needs to be faced. Some of the frameworks also do not support or hide the high complexity related to the configurations of big data environments which is another challenge for users that needs to be faced. Retrieving ML models to be used in production for other tasks without the need to build a new model for a new task is only supported by a few existing frameworks, namely Apache PredictionIO, ModelDB, Mistique and tensorflow serving. Caching of input data has the advantage of accelerating performing ML tasks, but it is also not supported in all of the frameworks presented in Table 3.1.

The challenge of processing large amounts of data in terms of execution time and overhead is tackled by using Big Data tools and a parallel distributed environment. However, such solutions are also only existing in a few of frameworks and summarized in Section 3.1.

Table 3.1.: Comparison of the data analytic and ML workflow management frameworks in related work to our framework.

Framework	Easy-to-use Web UI	Microservice Architecture	Support Big Data	Hide Big Data Complexity	Non-expert	Model Management & Retrieval	Caching	Generic	Distributed Processing	Automated Model Selection
Apache Spark [187]	-	-	+	-	-	-	-	+	+	-
Tesorflow [110]	-	-	+	-	-	-	-	+	+	-
Jupiter Notebook [73]	-	-	+	-	-	-	-	+	+	-
OceanTEA [60]	+	+	-	-	2A	-	-	-	-	-
WattDepot [26]	+	+	-	-	2A	-	-	-	-	-
Apache PredictionIO [33]	-	+	+	-	-	+	-	+	+	-
Microsoft Azure [36]	+	+	+	-	2A	-	-	+	+	-
ModelDB [166]	+	-	-	-	2A	+	-	+	+	-
Mistique [165]	+	-	-	-	2A	+	-	+	-	-
MLflow [185]	-	-	-	-	-	+	-	+	+	-
Tensorflow serving[106]	-	-	-	-	-	+	-	+	+	-
KubeFlow [23]	-	-	+	-	-	-	-	+	+	-
H2OFlow [27]	-	-	-	-	-	+	-	+	+	-
Our Contribution	+	+	+	+	**2A +2B**	+	+	+	+	+

A microservice-based architecture that has the main advantages of high scalability and easy maintainability of software solutions is found only in OceanTea, WattDepot, Apache PredictionIO and microsoft azure [36]. The other research projects presented in Table 3.1 are developed as monolithic solutions which make it difficult to scale and require the redeployment of the whole application when fixing bugs. While some of the discussed frameworks are developed as data analytics ones, the others are considered more as ML workflow management and visualization frameworks. None of them combined both categories in one conceptual framework to introduce more functionalities to the end users and facilitate performing ML tasks.

In this thesis, we tackle all of these challenges highlighting the novelty of our work by developing a conceptual microservice-based meta learning framework. The distinguishing factor of our conceptual framework lies in the fact that our framework combines all the advantages presented in Table 3.1 in one conceptual solution :

- Model managements: the framework implemented in this thesis tracks and stores the ML model and captures relevant metadata automatically for the user.
- It supports distributed processing, performing big data tasks and hides the low level high complex configurations of big data environments for the users.
- Supporting both categories of non-expert users by introducing highly configurable easy-to-use UI in which the user easily can manage and launch ML tasks on the cluster and by automating the process of selecting ML models.
- Storing and retrieving ML models to be used later for further tasks.
- Caching of input data to increase the efficiency in performing ML tasks in big data environments.
- The framework developed in this thesis is developed to be generic and able to handle a variety of ML tasks. However, energy load and generation forecasting scenarios are considered in the evaluation as will be seen later in this thesis.
- It combines both categories of software solutions namely, data analytics and ML workflow management and visualization frameworks in one microservice-based concept. This concept has the advantage of allowing plugging in several machine learning and deep learning runtime environments for future enhancements.

In order to support non-expert users category 2B in selecting the most appropriate ML model for a specific task, meta learning has been proposed to automate the process of model selection. State-of-the-art research projects in the field of meta learning to solve ASP are presented in Section 3.2. While some of them are proposed as concepts, the others are developed as framework solutions to select the best ML model without following the well-known trial-and-error approach.

Concerning meta learning concepts, a variety of meta features are extracted, namely simple, statistical, information-theoretic and landmarking as presented in Section 3.2. In terms of energy, for example, to solve ASP for a building, time series for building profiles besides the physical properties of the building are used as seen in [38]. The limitation of

this approach is related to privacy that leads to the non-availability of such properties in many applications situations. However, employing statistical properties as meta features is not new. Rossi et.al. in [131] used average, variance, minimum, maximum and median as meta features to develop a meta learning based method for periodic algorithm selection in time-changing data. The distinguishing factor in our thesis is that we extend the statistical meta features introduced in [131] to include more fine-grained as well as coarse-grained statistics in addition to the arithmetic mean as meta features known as Descriptive Statistics Time-based Meta Features (DSTMF) (see Chapter 5).

Our new set of meta features characterize energy time series datasets without having security as well as privacy issues. Better predictive performance is acquired by DSTMF compared to the cases existing in the literature. Another advantage and distinguishing factor of DSTMF lies in the complexity of extracting such meta features, whereby DSTMF introduces a smaller extraction time as well as overhead compared to the other meta features when they are extracted for a large amount of energy time series datasets.

To the best of our knowledge and as seen in all of the research projects presented in Section 3.2, there is no application of deep learning in the context of meta learning related to energy scenarios. It is clearly seen that only the original representation of meta features is utilized to describe time series datasets. In this thesis, we use unsupervised deep learning to encode the extracted meta features which can be seen as another novelty factor of our solution distinguishing it from the ones existing in the literature. To achieve that, we leverage the advantage of autoencoders to gain a deeper insight into the characteristics of time series datasets by encoding meta features into another more efficient representation form.

Concerning meta learning frameworks, such as Auto-WEKA, Auto-Sklearn, SmartML, most of them are developed as monolithic applications to select the best model in the context of classification. None of them is dedicated to perform automated model selection in the context of regression, for example, for energy load as well as generation forecasting. Moreover, they do not support ASP in big data environments. These challenges are tackled in our solution by presenting a microservice-based meta learning framework in Chapter 8. The distinguishing factor of our framework lies in the fact that it is built on top of a big data stack allowing it to perform automated model selection for large amounts of energy time series datasets. Another advantage of our solution over the existing ones presented in Section 3.1 lies in the utilization of a microservice architecture to achieve better scalability and maintainability.

Driven by the main definition of meta learning in which a meta learning problem is considered as a multi-class classification one, the size of the training sample on which the meta learner is trained, highly affects the predictive performance of the meta learning model. To enhance the predictive performance of multi-class classification problems by increasing the size of training examples, many solutions are proposed in the literature as seen in Section 3.3. While some of them generated new time series datasets by following model-based approaches in which new time series datasets are simulated, the others used genetic approaches to generate new training examples that are efficient and statistically in-differentiable from the original ones. The limitations of such approaches lies in the

complexity of building the models required to generate the new time series datasets. Also, such approaches often don't widen the diversity of the datasets because e.g. a model based approach follows a certain model which limits the diversity of the generated datasets if the model doesn't address all possible variations which are possible in real situations.

In this thesis, we propose two new approaches to generate new energy time series datasets. The aggregation of energy time series datasets has the advantages not only in DSTMF, but also it is proposed as a new approach to generate new energy time series datasets. Weather datasets are also used in our approach to generate new time series datasets based on the conditions of weather in the area in which time series datasets are collected. Beside the simplicity in calculation, our new approaches outperform model-based ones existing in the literature whereby the meta learning classification model introduced better predictive performance compared to the case of model-based approach as will be seen in Chapter 6.

4. Enhancing the Applicability of the Trial-and-Error Approach in Big Data Environments

Over the last decade, a variety of powerful algorithms and approaches for modeling and decision making from data are provided in Machine Learning (ML). Implementing a ML model is a complex, long lasting and error prone process whereby a large number of hyperparameter configurations need to be tried to find the best ML model. With the revolution of Big Data, where a large amount of data is generated and gathered each day, various software solutions are proposed to perform ML tasks, such as Apache Spark and Hadoop. Such solutions have the disadvantages of high complexity regarding the configurations of the underlying backends. Driven by that, the ML tasks are even harder to be performed by non-expert users.

In this chapter, we present our solution for supporting non-expert users with ML knowledge but without programming skills (category 2A, see Chapter 1) in performing ML tasks in Big Data environments. We start in Section 4.1 by clarifying the problem statement including motivations, challenges we aim to face and the scientific question we need to answer in this chapter. In this section, we precisely define the target group of non-expert users that are supported in our solution. In Section 4.2.1, we present our conceptual microservice-based architecture proposed to support non-expert users in performing ML tasks in Big Data environments. After that, the main execution workflow required to understand the main functionalities of our framework is explained.

We evaluate our work in Section 4.3 from four points of view. In the first one, the effect of chaching on the execution performance of our big data engine, namely Apache Spark is

Parts of this chapter are reproduced from:

- S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2019). "Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies". In: Proceedings of the 11th International Conference on Management of Digital EcoSystems, pp. 80–87. doi: 10.1145/3297662.3365807.
- S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies". In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XLV: Special Issue on Data Management and Knowledge Extraction in Digital Ecosystems, pp. 132–171. doi: 10.1007/978-3-662-62308-4_6.

investigated. The major advantage of our framework in storing the pre-trained ML model to be used later for new tasks is evaluated in terms of time. Thereafter, the efficiency of our microservice-based framework in terms of overhead is also evaluated for different sizes of energy time series datasets. To precisely define the best practice in using our framework, we define and evaluate the required thresholds and conditions, at which it is highly recommended to use big data environments in favor of single computers for performing a given ML task. The research contributions presented in this chapter were the main topics of our papers in [141][142].

4.1. Problem Statement

Besides the advantage of ML in solving many complex business problems, there are also some downsides. It is a time-consuming process for the user to apply ML according to the well-known trial-and-error approach whereby a lot of model hyperparameters need to be configured to achieve the best performance. Such an approach is based on the idea that all possible combinations of learning algorithms with their relevant parameters are tried for each task until a good solution is found. Consequently, it wastes the resources for constructing multiple models which can take a long time especially in the context of Big Data.

In order to build an ML model, several steps including data preprocessing, splitting the data into training and test data, model training and model testing are required. Such process is more difficult in the context of Big Data where a large amount of data need to be processed. With the increasing amount of available data, various libraries and systems have been introduced to enable large-scale distributed/parallel processing. One of the best known open-source frameworks is Apache Spark [187], Hadoop Distributed File System (HDFS) [149] and Yet Another Resource Negotiator (YARN) [167], to name a few. Apache Spark supports Big Data processing and storage in a distributed environment. It encompasses various components including a distributed file system, the data processing tool MapReduce and a cluster resource manager. HDFS enables the reliable storage of extensive files in a cluster. It provides fault tolerance by splitting the files into blocks and replicating these blocks multiple times over the cluster. YARN is a technology that decouples the application and the required computational resources (e.g. CPUs, RAM, etc.) for processing from the resource management infrastructure of the cluster. Besides the advantages of such technologies, they do have some drawbacks for non-expert users. It will be difficult or even impossible for them to execute ML tasks in Big Data environments. This is due to the lack of experience needed to set up such tasks, and the required configurations to successfully submit and execute jobs in Big Data environments.

In this chapter, we answer the research question RQ1 presented in Section 1.2. We supported non-expert users category 2A presented in Table 1.1 in performing ML tasks in Big Data environments. We developed a new microservice-based solution helping the aforementioned non-expert users to solve ML problems in Big Data environments without caring too much about technical issues of the underlying Big Data and cluster computing

environment as a runtime platform. Our solution facilitates training, testing, managing, storing and retrieving ML models. It is provided with an easy-to-use highly configurable UI in which the model and the required hyperparameter can be set.

To evaluate our concept, the short-term (hourly) power generation forecasting scenario is taken into consideration. In the following sections, the conceptual architecture of the proposed framework, the execution workflow, evaluation and the experimental results are explained in detail.

4.2. Proposed Solution

In this section, we present our conceptual microservice-based architecture proposed to answer the aforementioned research question RQ1. To this end, we clarify in detail the layers, the microservices and the communication between them to achieve the main goal of our framework. For better understanding of the functionalities of the different layers involved in our microservice-based framework, the general execution workflow is presented and explained in detail in Section 4.2.2.

4.2.1. Conceptual Microservice-Based Architecture

Figure 4.1 illustrates the conceptual architecture of the presented framework. As seen in this figure, the architecture consists of three main layers, namely the UI layer, service layer and persistence and processing layer. The UI is split into separate sub-parts (e.g. separate web applications) providing dedicated functionalities for data and model management, model training and cluster management which are wrapped into one logical web application forming the UI of the application. Two microservices are incorporated together to perform the service layer. Each service is a small and self-contained application that can be deployed independently e.g. on the runtime cluster with a single responsibility. While one service is responsible for data and model management, where models can be seen as special data objects, the other service focuses on the management of running ML jobs e.g. for training and testing.

To allow web applications in the UI layer to interact with the runtime environment, the services provide RESTful APIs. The persistence and processing layer provides the basic model and data storage capabilities according to the underlying runtime computer infrastructure. Moreover it provides generic interfaces for running and managing ML jobs on this infrastructure independent of the used low level ML framework.

4.2.1.1. User Interface (UI) Layer

As seen in the architecture presented in Figure 4.1, the User Interface (UI) layer consists of separate web applications providing the dedicated functionalities of the framework.

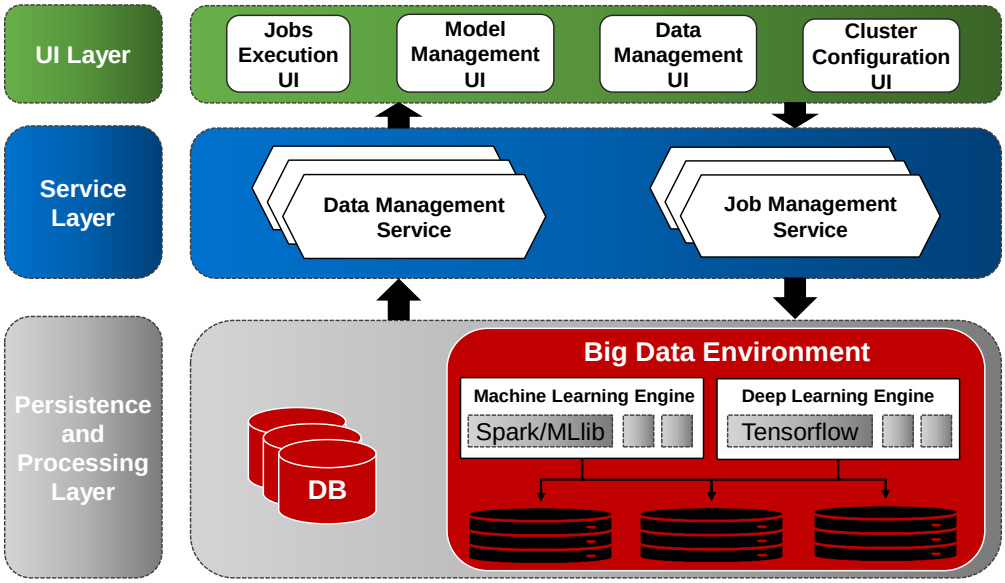


Figure 4.1.: Basic architecture of the proposed microservice-based framework [141][142].

These applications interact with the service layer via RESTful APIs and are wrapped into a container application which provides navigation between the views to form the complete UI. To make the user experience of the UI as pleasant as possible, the famous 10 Usability Heuristics for UI Design by Nielsen [102][103][57] are applied while conceptualizing and implementing the UI. Multiple technologies including HTML5, CSS and React are utilized to implement the UI. React [152], the JavaScript (JS) library from facebook, is chosen because it simplifies the development of complex user interfaces. Its good performance can be attributed to its use of a virtual Document Object Model(DOM) which is a copy of the HTML DOM and enables efficient rendering updates of the otherwise slow HTML DOM. React is based on declarative programming and the concept of encapsulating and reusing of components. Such components are implemented through a specific syntax called JavaScript Syntax Extension (JSX) which is a combination of HTML and JS code.

In this UI, Node Package Manager (NPM) is used to simplify the configuration of the build tools and the setup of React application. For better data management and to organize the side effects related to asynchronous RESTful API calls, Redux and redux-saga are used [21]. To distinguish different functions and to provide good navigability on the website, React Router is utilized. For implementing a responsive and nice web design, the popular framework React Bootstrap [154] which provides easy to use pre-styled components is utilized.

A recent trend in web development has been to develop web UIs as Single Page Applications (SPAs) [96]. Essentially, SPAs are front-end applications that consist of a single HTML document that can be dynamically updated through JavaScript (JS). This makes it possible to refresh only particular regions of the screen instead of reloading the whole page when changes take place. This is especially convenient in interactive web pages, since these applications can respond much faster to user input and therefore provide better user

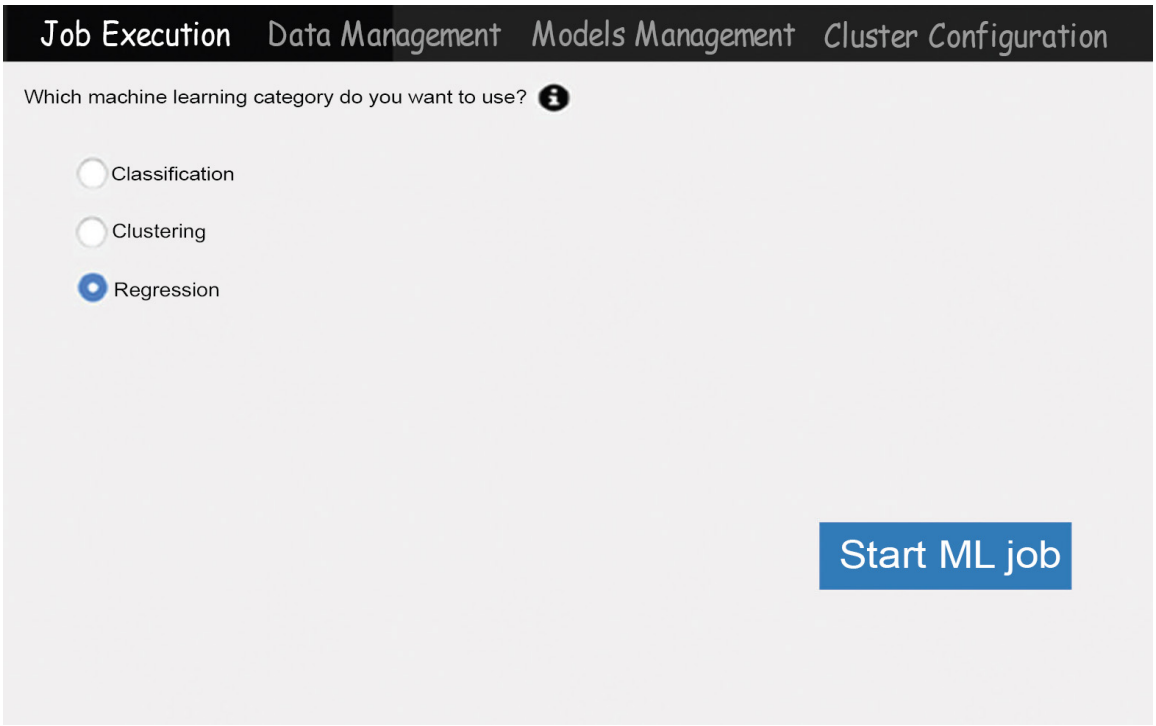
experience. Additionally, the number of requests between the SPAs and services is often dramatically decreased, since much of the logic can be implemented in the front-end. For these reasons, the web UI is implemented as an SPA communicating with the service tier through HTTP requests using the RESTful APIs. In the current version of the concept, the UI contains separate web applications for "data management", "model management", "execution of jobs" (e.g. for training and testing) and "cluster management". Figures 4.2 and 4.3 show some web page views related to these applications. In following, the main UIs in the UI layer are presented in more detail.

Data Management UI: It is responsible for uploading, managing and configuring data sources which provide data to ML jobs. To understand the different characteristics and properties of input datasets, an interactive visualization and statistical analysis can be performed in this UI. For example, in the case of time series datasets, the user has the ability to zoom in/out and select a part of the chart for more detailed view. This allows the user to discover trends and outliers in the selected part of the time series dataset. Additionally, when the user hovers over a specific point in the chart, the related information will appear in a small box, for example the value of the power generation at this point. The interactive visualization of statistics and performance data in our framework is implemented using the HighChart Java-script library [42].

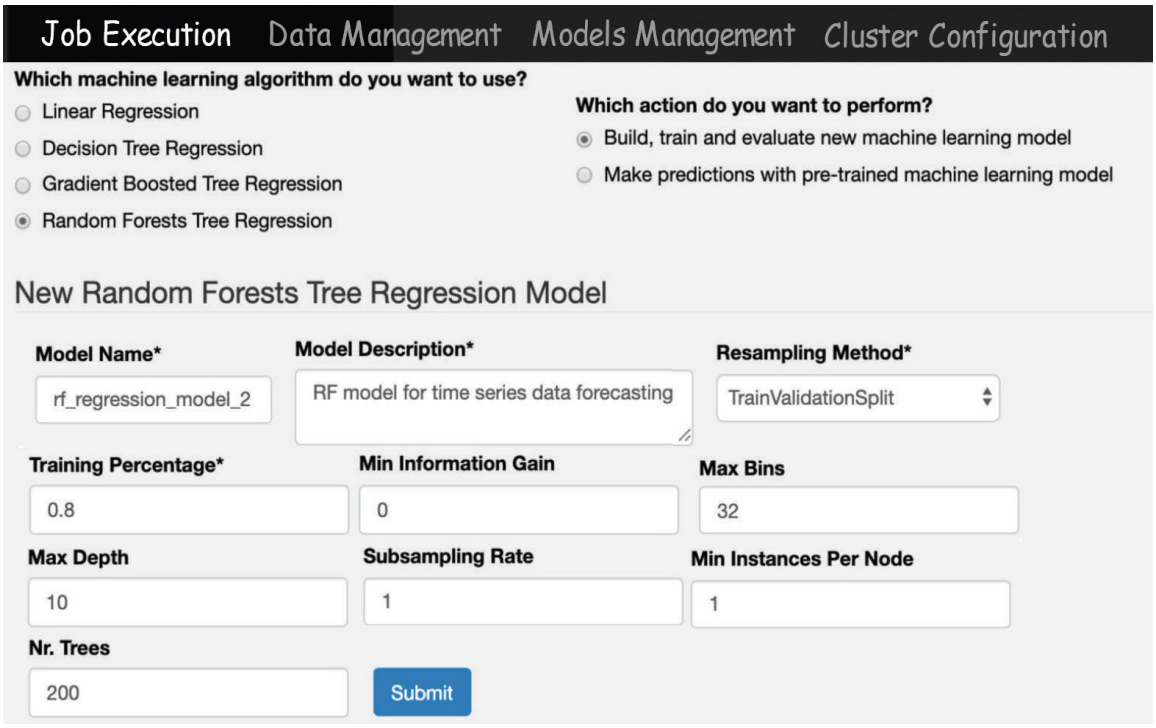
Job Execution UI: As mentioned before, the main goal of our microservice-based framework is to support non-expert users in performing ML tasks in Big Data environments. To this end, this UI provides functionalities for executing a job for training and testing a ML model. An example for ML job can be the building of a model for load forecasting. To ease the usage for non-experts category 2A presented in Table 1.1, the UI provides a wizard interface which guides the user through the process of choosing a dataset, a type of analysis to be performed on the dataset, an adequate ML model (e.g. model, either pre-trained or untrained) for performing the wanted type of analysis and afterwards for tuning the execution parameters of the model based on an already existing parameter set.

One of the main advantages of the proposed framework is to be very generic. I.e., in the step of selecting a given type of analysis to be performed on a dataset, the user should be able to select many different types of ML based analysis. But what kind of ML analysis methods and algorithms will be available is directly dependent on what kind of low level ML frameworks will be integrated on the persistence and processing layer. Because in the present work only Apache Spark is integrated as a low level ML framework and Apache Sparks standard ML library mainly provides algorithms for classification, clustering and regression, our framework currently only provides these three categories for choosing an analysis category as shown in Figure 4.2a.

After choosing one of these categories, the user is navigated to the datasets tab view in order to select an already uploaded dataset or data source, or directly upload one to perform the ML task. Thereafter, the wizard navigates to the next wizard screen shown in Figure 4.2b. This figure shows that a ML framework can provide a variety of ML algorithms for performing a certain analysis category to cover a wide range of ML application scenarios. I.e., it can be seen in Figure 4.2b that Apache Spark provides several



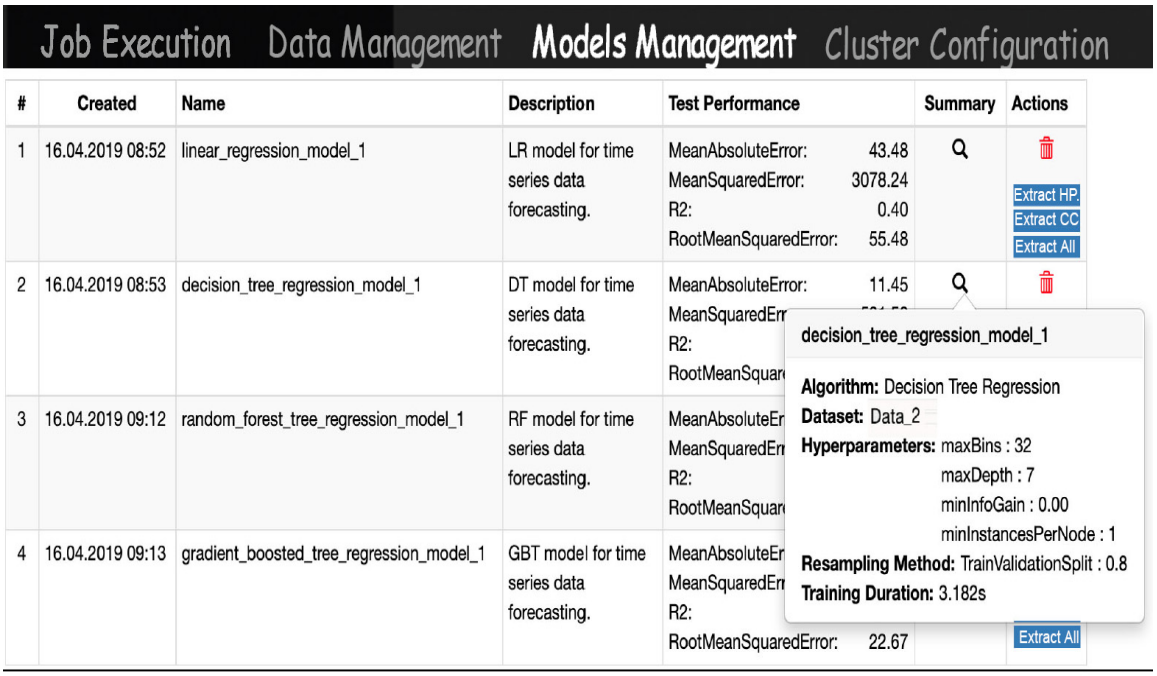
(a) Job Execution UI - Choosing ML Category



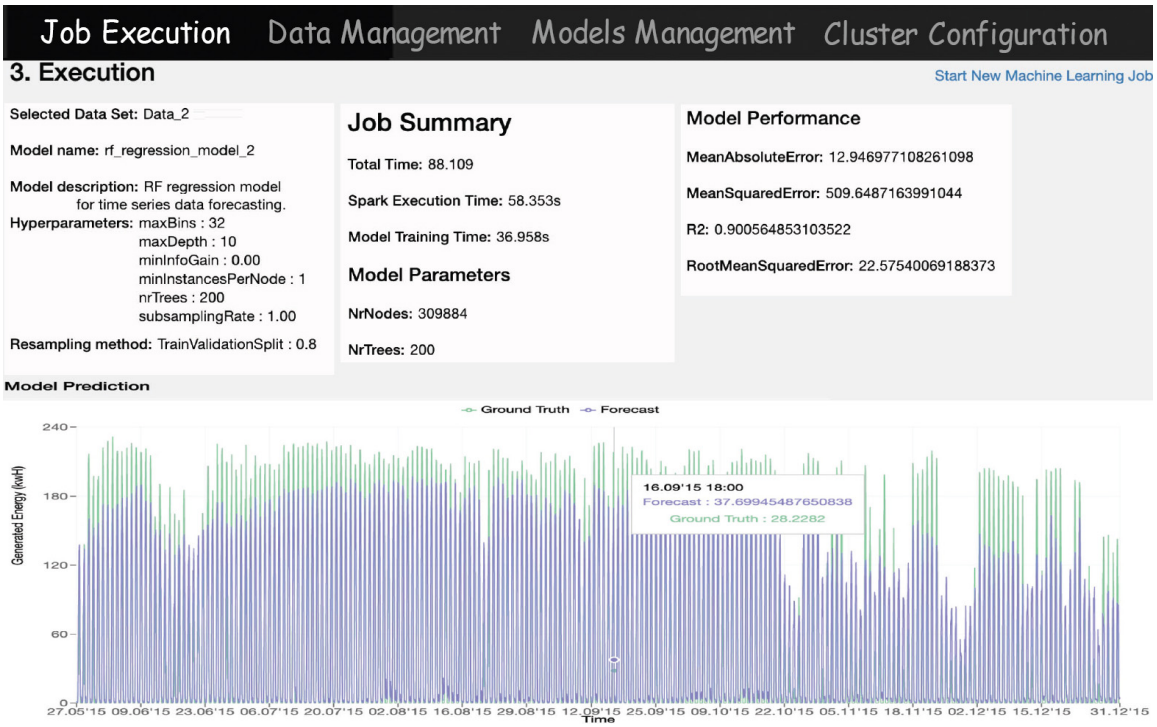
(b) Job Execution UI - Building ML Model

Figure 4.2.: User Interface (UI).

algorithms for “regression analysis”, e.g. “linear regression”, “decision tree regression” and



(a) Model Management UI



(b) Job Execution UI - Summary

Figure 4.3.: User Interface (UI).

so on. If at a later time more than one ML framework is incorporated into the present

framework, different algorithms implementing another analysis category can even be provided.

Another advantage of our framework is the storage and re-usability of pre-trained ML models on new datasets. This reduces the time needed by the user to solve his problem for a new dataset. It can be seen from Figure 4.2b, that the user has the possibility to use an already existing pre-trained model or alternatively create and train a new ML model. Here, the user can adapt a given collection of algorithm hyper-parameters for tuning the model performance. After appropriate options are chosen in Figure 4.2b, the ML task including learning and testing can be executed on the runtime platform. The wizard will then show a screen which allows to monitor the execution state. When the execution is done, the model and the other results of execution are saved in the persistence and processing layer and a comprehensive visualization of results as well as an execution summary are shown as depicted in Figure 4.3b.

Model Management UI: As its name implies, the model management UI is responsible for managing ML models which are pre-trained in the framework. Figure 4.3a shows a view of this UI which lists the available models. Each model is described with some associated metadata (e.g. id, creation date, model name, a textual description of what the model does, etc.) which are shown in the tabular view. Each row (e.g. a pre-trained model) represents a ML pipeline corresponding to a specific ML task. For each task, the related general information resulting from performing this task such as ML algorithm, dataset used for training and testing, hyperparameters and performance results, to name a few, are shown if the user hovers over the model entry in the model list. To this end, the user can compare models and select the best one for executing it on a new dataset. Moreover, the user can perform actions on a selected model, namely delete a pipeline, extract the best hyperparameters, extract cluster configurations or extract the whole parameters and use them to build a new ML model.

Cluster Configuration UI: As mentioned in the introduction, a Big Data infrastructure as a runtime environment for ML tasks can introduce great challenges for configuring and running the framework on the cluster with best performance for a given task. To tackle this challenge, the cluster configuration UI implemented in this framework gives the possibility to tune the low level execution framework configurations in relation to the usage of CPU cores, RAM usage and executors instances, to name a few.

4.2.1.2. Service layer

In this layer, the generic interfaces to the runtime environment are provided via currently two microservices, namely the Job Management Service (J.M.-Service) and the Data Management Service (D.M.-Service) as shown in Figure 4.1. These interfaces are accessed by UI applications to setup and execute ML jobs in Big Data environments. Each microservice has dedicated responsibilities and contains a layered architecture based on the Separation of Concerns (SoC) design principle. Keeping the code in distinct layers enforces a logical encapsulation of functionalities and dependencies leading to better code maintainability

and loose coupling. Figure 4.4 depicts this architecture, where only upper layers are allowed to access lower layers.

To handle HTTP requests and form the entry point of the microservices, the presentation layer is provided. It contains controllers which map HTTP URLs and provide Create, Read, Update and Delete (CRUD) functionality to the outside through RESTful APIs. For simple read requests, the layer accesses the persistence layer to acquire the relevant data from the database. However, for complex logic, it communicates with the service layer which contains the business logic. This has the advantage that common operations required by multiple controllers can be abstracted to the service layer. The persistence (i.e. data access) layer consists of repositories and entities. The repositories interact with the underlying data source i.e. database and manage the entities which encapsulate the domain objects.

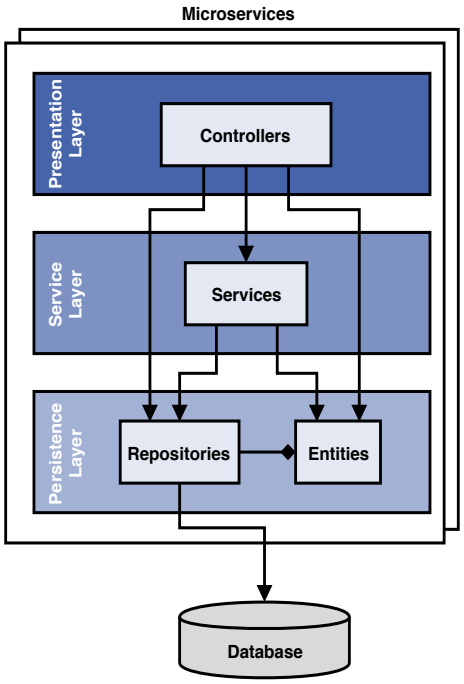


Figure 4.4.: Layered architecture microservice.

A comprehensive description of both microservices, which are called as services for a simplicity's sake is provided in the following two sections. The established RESTful pattern is chosen as the communication tool instead of the event-driven pattern, because the microservices are just two in total and the RESTful communication is easier to implement. In addition, the JSON format is selected for requesting and sending data via the RESTful APIs because of its popularity, ease of use and interpretability.

J.M.-Service: To create and submit ML jobs to be executed by an available low level ML execution framework (e.g. Apache Spark) on the available runtime environment (e.g. a cluster or single computer), this service is developed. This job includes data preprocessing and building forecasting models, to name a few. It interfaces with the persistence and processing layer below which encapsulates the specification of a certain runtime environment. For better execution and managing of ML tasks, the J.M.-Service is

Table 4.1.: List of the URL patterns of the J.M.-Service.

URL Pattern	Description
/jobs	A GET request on this URL is used a list of spark jobs
/jobs	A POST request on this URL is used to create of a spark job and its corresponding processing directory in HDFS
/jobs/id	A GET request on this URL is used to retrieve a spark job for a specific id
/jobs/id	A DELETE request on this URL is used to delete a spark job for a specific id with its corresponding processing directory in HDFS
/jobConfigurations	A GET request on this URL is used to show a list of spark configurations
/jobConfigurations	A POST request on this URL is used to create spark configuration
/jobConfigurations/id	A DELETE request on this URL is used to delete a specific spark configurations for specific id
/jobSetup	A POST request on this URL is used to copy the packaged jars and pre-trained saved machine learning models into HDFS
/submitJob/id	A POST request on this URL is used to submit a spark job

not only responsible for executing ML tasks but also tracking and monitoring the status of the running tasks.

Moreover, it reads the execution results stored by the executing framework somewhere in the runtime environment (e.g. in an execution directory of the task on e.g. a file system) and sends them to the D.M.-Service for storage in a database, so that the execution statistics and results can later be visualized in the UI. The J.M.-Service provides an abstract job execution and monitoring interface to the web application UI through its RESTful APIs. This completely decouples the UI from the specification of the runtime environment. The main functionalities of J.M.-Service REST-APIs are summarized by the URL patterns presented in Table 4.1.

D.M.-Service: To perform ML tasks, the required input datasets and ML algorithms as well as scripts need to be available and prepared. This service is responsible for the storage and preparation of required inputs to execute a job on the runtime environment, namely storing and providing datasets, models containing (pre-trained) algorithms and hyperparameters, to name a few. To store the required data and the results produced from performing ML tasks, the D.M.-Service uses its own database. On the one hand, the UI applications interact with this service to upload, manage and retrieve data, model information as well as configurations. On the other hand, the J.M.-Service interacts with the D.M.-Service to retrieve information about datasets, models and configurations. J.M.-Service service is also responsible for copying models from the database to the execution environment of a task and pushing results back to the D.M.-Service. The D.M-Service then stores all information about the execution of a task and the results in its own database, so that these information can be later used for the visualization of the results and the overall performance of the ML jobs as already shown in Figure 4.3b.

Table 4.2.: List of the URL patterns of the D.M.-Service.

URL Pattern	Description
/algorithms	A GET request on this URL is used to retrieve a list of the available machine learning algorithms
/algorithms/id	A GET request on this URL is used to retrieve a specific machine learning algorithm
/categories	A GET request on this URL is used to retrieve a list of the available machine learning categories, for example classification, regression, clustering, to name a few
/dataSets	A GET request on this URL is used to show available datasets
/dataSets	A POST request on this URL is used to create meta data of a dataset
/dataSets/id	A GET request on this URL is used to retrieve the metadata of a specific dataset
/dataSets/id/data	A POST request on this URL is used to upload a local data file into HDFS and upload the dataset's reference
/dataSets/id/descriptiveStatistics	A POST request on this URL is used to prepare model for calculating the descriptive statistics for a specific dataset
/mlModels	A GET request on this URL is used to retrieve a list of pre-trained machine learning models
/mlModels/id	A GET request on this URL is used to retrieve metadata of a specific machine learning model
/mlModels/id	A DELETE request on this URL is used to delete a specific pre-trained machine learning model
/mlModelPredictions/id	A GET request on this URL is used to retrieve the predictionfile for a specific machine learning model
/mlPipelines	A GET request on this URL is used to retrieve a list of machine learning execution pipelines
/mlPipelines/id	A GET request on this URL is used to get the meta data for a specific machine learning pipeline

The main functionalities of the D.M.-Service REST-APIs are summarized by the URL patterns presented in Table 4.2.

4.2.1.3. Persistence and Processing Layer

Hiding the low level details of the runtime environment from the implementation of the services is one of the major advantages of this layer. Both J.M.-Service and D.M.-Service uses generic functions implemented in this layer to interface with the job runtime directory in HDFS and the database infrastructure installed on the runtime as well as performing dedicated tasks on the runtime environment for instrumenting installed ML frameworks to e.g. perform job execution. For each ML runtime environment, the persistence and processing layer will contain an adapter which maps model and execution details to the specific framework (see Section 4.2.2 for further discussion on issues related to the prototype and interfacing to the Apache Spark runtime environment).

Typically, a job runtime directory is created in the file system of the runtime platform to store all information related to the execution of a certain job. To achieve that, the persistence and processing layer contains functionalities for creating such directories depending on the execution framework. More generally, all data items managed by the D.M.-Service are stored in a database infrastructure which is defined by an abstract object-like interface. This interface can be implemented in the runtime infrastructure by using different database technologies as shown in the next section.

4.2.2. Execution Workflow

The previous sections elaborated on the architecture and the design of the framework. The focus now is shifted to how the main components communicate with each other to achieve the main goals of the framework, that is, assisting non-expert users in training, testing and managing ML tasks in Big Data environments. To this end, we introduce the basic execution workflow necessary to understand the main functionalities of our framework.

Apache Spark as one of the state-of-the-art Big Data processing environment is installed on a Big Data computing cluster using an Apache Hadoop software stack as runtime engine for executing ML jobs. ML execution environments typically use a job runtime directory in a file system for storing all information needed for job execution (e.g. for storing models to be executed, algorithm configurations and results). On a Big Data cluster based on Apache Hadoop, HDFS is typically used as a distributed file system and the runtime directory for a job can be accessed by all computing nodes of the cluster using the HDFS interfaces. Therefore, for implementing the persistence and processing layer on the cluster, HDFS and a postgreSQL [105] database are utilized to store the required input and the output produced from performing ML tasks. The postgreSQL database system is used as an object-relational database to store all information managed by the D.M.-Service, e.g. ML categories, ML algorithms, hyperparameters, pre-trained models, jar files, references of datasets stored in HDFS, pre-trained model pipelines and untrained model pipelines. The main difference between pre-trained and untrained model pipelines is explained in detail in Section 4.3.

To store datasets and the output of successful jobs executed in Apache Spark before being read by the J.M.-Service, HDFS is also utilized. The dataset storage on HDFS allows it to have “Big Data” as input, i.e., datasets which are extremely large. To achieve the goal of storing pre-trained ML models in the form of binary objects, the Large Object feature of PostgreSQL is used. This feature uses the Large Object Manager Interface which stores only a reference named oid in the database table pointing to the actual object stored in the system table pg_largeobject. This method breaks the binary data into chunks and allows storing objects of up to 2GB within the database. However, another format such as the Predictive Model Markup Language (PMML) will be considered in the future.

Figure 4.5 shows the basic methodological workflow for task execution as it is implemented in the prototype for submitting jobs to the Apache Spark runtime. For each new job, the persistence and processing layer generates on behalf of the J.M-Service a Universally

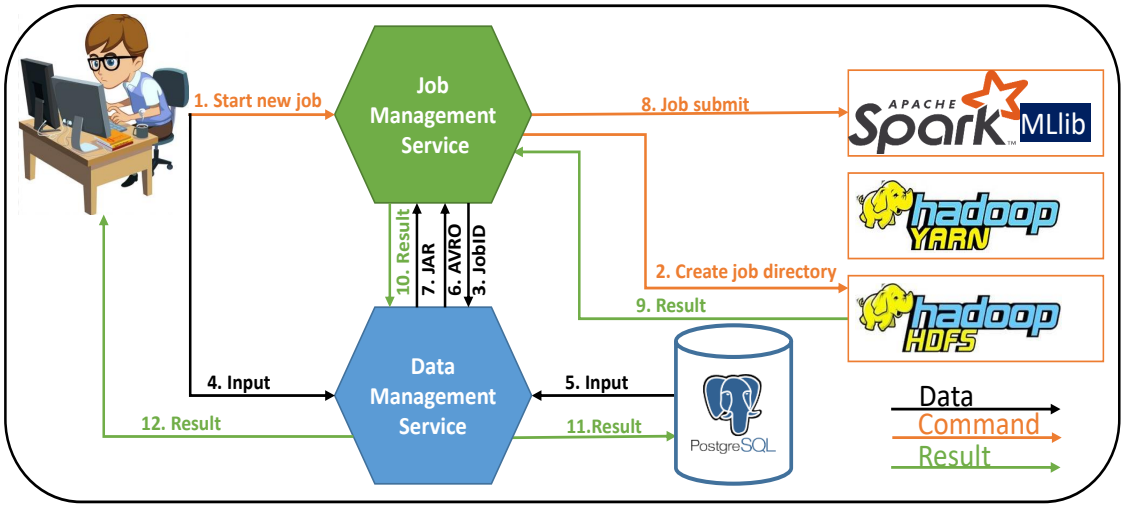


Figure 4.5.: Execution workflow.

Unique Identifier (UUID) as jobID which is sent back to the D.M.-Service. The usage of a UUID guarantees the uniqueness of the id, making it suitable to use in a distributed environment, such as a Big Data environment.

A temporary job runtime directory with the UUID as a name is created in HDFS by the J.M.-Service for each jobID. To this end, the J.M.-Service uses the File System (FS) shell instruction of HDFS . After that, the J.M-Service calls the D.M.-Service to fetch the necessary artifacts (e.g. model, runtime configurations) from the database and pass it to the J.M-Service as an Apache Spark AVRO file. After that, the J.M-Service places the AVRO file in the persistence and processing layer in the job runtime directory.

The decision for utilizing AVRO was made, because AVRO uses a schema that decouples the solution from the implementation including error prevention. An AVRO file contains the received jobID and the chosen cluster configurations. However, if no cluster configurations are chosen in the UI, the default one are fetched from the database and used in this task. Besides cluster configurations, algorithm hyperparameters and metadata related to the execution of algorithms, namely the name of the application main class are included in the AVRO file for execution. The name of the application main class is required by Apache Spark to find the main code entry point for executing the task. While all datasets are stored in the HDFS, path references pointing to the files are stored in the database of the D.M.-Service.

Once the user chooses a dataset, the path reference of the dataset in HDFS is fetched from the database and included in the AVRO file. After that, the D.M.-Service fetches the corresponding jar file from the database and sends it to the J.M.-Service. At this point, all required information to perform the task is passed to the J.M.-Service which creates a spark-submit job and sends it for execution to Apache Spark.

https://hadoop.apache.org/docs/stable/

As a result of executing e.g. a task performing forecasting on a time series dataset, the forecasting results, forecasting performance and the forecasting model in the form of a binary object are located in the temporary job runtime directory of the task. After executing the job, all of these results are stored in the temporary job runtime directory and read afterwards by the J.M.-Service to be passed to the D.M.-Service. The D.M.-Service receives the results and stores them in the form of a pipeline in the database to be retrieved later. Simultaneously, the D.M.-Service sends the results to the UI to be rendered and visualized for the user.

4.3. Evaluation

In this section, four aspects related to the evaluation of our microservice-based framework are investigated. We start by analysing and discovering the effect of caching in Apache Spark. In this context, we compare the execution time of training and testing the benchmark evaluation models in case of memory caching and without memory caching of the input time series datasets. Moreover, the execution time and framework overhead for evaluating the efficiency of the framework are measured, highlighting the advantage of storing and retrieving ML models and discovering the threshold, at which the use of the proposed framework is recommended for better performing machine learning tasks in Big Data environments. In these experiments, datasets presented in 2.4.2 and 2.4.3 are used. Before presenting the obtained results, the experimental setup and the related configurations are presented.

4.3.1. Experimental Setup and Configurations

We implemented the aforementioned microservice architecture using Java on a local workstation which is a MacBook with a 2.7 GHz Intel Core i5 processor and 8GB of RAM. Both microservices are implemented as standalone Spring Boot applications which are configured to run on different HTTP ports, namely 8090 and 8080. To run our web application, the embedded Apache Tomcat server from Spring Boot is utilized.

To precisely investigate the effectiveness of the framework, a local execution context and cluster execution context have been configured and used in our experiments. In the local context, Spark (v. 2.3.0) on top of Hadoop (v. 2.7.6) as state-of-the-art technologies to perform ML tasks is installed on the aforementioned workstation, where the executors and drivers run in a single JVM. In the cluster context, we utilize a powerful Big Data stack, in which Apache Spark is fit on top of Yet Another Resource Negotiator (YARN) as a resource manager and Hadoop Distributed File System (HDFS) as a primary data storage. The Big Data stack is deployed on a cluster of 3 logical machine nodes. Each of them has 32 cores and 80.52 GB RAM. The nodes are connected to each other by a LAN with 10 GBit/s bandwidth.

Table 4.3.: Default and custom cluster configurations used in cluster context.

Default	Custom
Drivers.cores = 1	Drivers.cores = 1
Driver.memory = 1 GB	Driver.memory = 1 GB
Executors.cores = 2	Executors.cores = 2
Executors.memory = 1 GB	Executors.memory = 70 GB
Executors.instances = 1	Executors.instances = 3

To tune an application's performance, Spark provides an abundance of configuration parameters. A focus of this evaluation is on adjusting the cluster's main resources, the CPUs and amount of RAM, used by the applications, since they can greatly impact the computational performance. In the cluster context, we distinguish two configuration setups, namely default and custom including driver cores, driver memory, executor cores, executors memory and executors instances as presented in Table 4.3.

Random Forest (RF) [140], Multiple Linear Regression(MLR) [100], Gradient Boosted Trees (GBTs) [115] and Decision Tree (DT) [161] are used as base classifiers to build the data-driven forecasting models. The focus is on forecasting the hourly generated power of the solar PV systems using the weather condition and time features. MLlib, which is a Spark scalable ML library, is employed to build the models. To train and test the forecasting models, Ausgrid solar home electricity data presented in Section 2.4.2 is used. MLR is a widely used supervised algorithm which assumes a linear relationship between one or multiple independent input variables and a dependent output variable. Table 4.4 presents the default values of the MLR hyperparameters.

Table 4.4.: Default hyperparameters of MLR algorithm in MLlib.

Hyperparameter	Description	Default
MaxIter	Maximum number of iterations	100
RegParam	Regularization/Shrinkage parameter	0.0

The DT algorithm is a supervised ML algorithm that has the ability to capture the non-linear structures in data. It is based on the idea of building a binary tree which recursively partitions the input space and consists of internal nodes and leaves (i.e. terminal nodes). It is constructed starting from the root and its nodes are split down based on the largest decrease in impurity. For classification trees, the impurity is often measured with the Gini impurity or entropy. However, for regression trees, where the target is continuous, the impurity is based on variance reduction. Table 4.5 presents the default values of the DT hyperparameters.

https://spark.apache.org/docs/latest/ml-guide.html

Table 4.5.: Default hyperparameters of DT algorithm in MLlib.

Hyperparameter	Description	Default
MaxBins	Maximum number of bins for split decision and discretization of continuous features	32
MaxDepth	Number of trees in the forest	5
MinInstancesPerNode	Minimum number of trees (training instances) in children must have by splitting	1

A forest of multiple DTs is built in RF algorithm. Each DT is trained independently. While single DTs are often said to overfit, the RF algorithm does not overfit because of the law of large numbers [53]. Also, randomness is applied to the training process of RF by utilizing random feature subsets for node splitting. Since, each DT is trained separately, multiple trees can be trained in parallel. For the final prediction, the individual votes of all trees are combined. Table 4.6 presents the default values of the RF hyperparameters.

Table 4.6.: Default hyperparameters of RF algorithm in MLlib.

Hyperparameter	Description	Default
MaxDepth	Maximum depth of individual trees in the forest	5
NumTree	Number of trees in the forest	20

In contrast to RF which trains the trees independently, GBTs algorithm employs the Boosting technique training one tree at a time. Successively, to correct the errors made by previous trees, a DT is fitted on the residuals of the previous tree, instead of a fraction of the original data. The final prediction is based on a weighted majority vote. Table 4.7 presents the default values of the GBTs hyperparameters.

Table 4.7.: Default hyperparameters of GBTs algorithm in MLlib.

Hyperparameter	Description	Default
MaxDepth	Maximum depth of individual trees in the forest	5
MaxIter	Maximum number of iterations	20
StepSize	Controls the contribution/weight of each tree	0.1
SubsamplingRate	Training data proportion used for learning each tree	1.0

Typically, tuning hyperparameters is an important step of the Machine Learning Pipeline (MLP), since they can not only significantly influence the forecasting performance of a model, which is not our focus in the present work, but also the processing time.

Based on the main property of our microservice-based framework in facilitating training and testing ML models in Big Data environments, an efficient hyperparameter tuning is

Table 4.8.: ML algorithms hyperparameters after tuning.

ML Algorithm	Hyperparameters
Multiple Linear Regression (MLR)	Max iterations (ntree) = 20 Regularization parameter = 0.5
Decision Tree (DT)	Max bin = 5 Max depth = 5 Min instance split = 1
Gradient Boosted Trees (GBTs)	Max depth = 5 Number of trees = 20 Step size = 0.1 Sampling rate = 1.0
Random Forest (RF)	Max depth = 5 Number of trees (ntree) = 20

performed for the aforementioned ML algorithms to ensure that the time measurements are taken for a best case scenario of the aforementioned algorithms. The results are depicted in Table 4.8.

One of the main advantages of the proposed framework is the ability to store pre-trained models in order to use them later in production. Thus, for evaluation, two execution contexts are determined, namely the untrained model pipeline and pre-trained model pipeline. In the first one, as its name implies, the user follows the general methodology to perform an ML task, in which the model is trained from scratch and afterwards tested. In the second one, the user selects a pre-trained model from the database and uses it to perform or test an ML task with a new dataset without the need for building a new model. As mentioned before, the main goals of evaluation are discovering the effect of caching in Apache Spark, the advantage of storing ML models and reusing them, measuring the framework overhead and determining the thresholds for efficiently performing ML tasks on the cluster. To this end, time measurements need to be precisely defined. As time measurements, we defined T_{total} and T_{fo} according to Equation (4.1) and (4.2) respectively.

$$T_{total} = T_{st} + T_{fo} \tag{4.1}$$

where:

- T_{st}: is the execution time required by Apache Spark to perform a ML task in context of pre-trained pipelines or untrained pipelines.
- T_{fo}: is the framework overhead.

$$T_{fo} = T_{co} + T_{dbo} \tag{4.2}$$

where:

- T_{co}: describes the communication overhead between microservices and inside the Big Data infrastructure.
- T_{dbo}: describes the overhead for storing and retrieving required data from the database.

4.3.2. Results and Discussion

So far the experimental setup and the configurations required to perform ML tasks on clusters are presented. In the following, the evaluation results are discussed. As the focus lies on the execution time and the framework overhead raised while performing ML tasks, the accuracy of forecasting will not be taken into account.

4.3.2.1. Effect of Caching in Apache Spark

The basic data structures of Apache Spark, namely Resilient Distributed Datasets (RDDs), are developed as a fault-tolerant immutable collection of data objects and can be computed on different nodes of the cluster [186]. In this context and in order to speed up the running applications, caching RDDs in Apache Spark plays an essential role. This is especially helpful, when running iterative ML applications, where the data is accessed repeatedly. If RDD is not cached, nor check-pointed, it is re-evaluated again each time an action is invoked on that RDD. In our evaluation, the training time is measured as the time it takes to fit the model on the training data. The prediction time is similarly computed for applying the resulted model on testing data. Since Spark utilizes lazy evaluation for data transformations, meaning an operation is not executed until an action is called on the data, the prediction time has to be measured in combination with performing an action. The main advantages of the lazy evaluation mechanism in Apache Spark are:

- Increased manageability of RDDs because the source code of our machine learning algorithms is organized into smaller operations which in turns reduces the number of passes on data by grouping the operations.
- More efficient computation time and an increased speed, as only the necessary values are computed saving the communication round-trip time between the drivers and clusters.
- Better optimization of operations on data by reducing the number of queries.

Table 4.9 shows how the caching of the input time series datasets affects the performance of the implemented algorithms, using their default hyperparameters and default cluster configurations. For calculating these values, the experiments are repeated three times. Afterwards, the mean values are calculated as final performance indicators. Obviously, the need for caching is larger in the case of large datasets, as more operations are required and larger amounts of data are loaded and accessed repeatedly, therefore and to precisely discover the effect of caching, the models are trained and tested on a small dataset size i.e.,

Table 4.9.: Mean computation time for training and testing different algorithms in the cases of caching and no caching of input data.

Machine Learning Algorithms	Training Time (s)		Prediction Time (s)	
	No Caching	Caching	No Caching	Caching
Multiple Linear Regression (MLR)	16.07	3.57	3.92	0.87
Decision Tree (DT)	15.88	3.21	3.41	0.86
Gradient-boosted trees (GBTs)	37.04	21.74	8.61	1.77
Random Forest (RF)	23.11	12.48	5.75	1.12

4 MB. As shown in this table, combining lazy evaluation with caching reduces the training and prediction time of all algorithms by approximately 75%.

4.3.2.2. Advantage of Storing and Retrieving ML Models

In the second group of our experiments, we evaluate the advantage of storing and retrieving ML models. The main ML task is to perform short-term power generation forecasting using MLR, RF, DT and GBTs data-driven models on the Ausgrid solar home electricity data presented in Section 2.4.2. The weather dataset presented in 2.4.3 is used to extract the required features for building forecasting models. The algorithm hyperparameter configurations shown in Table 4.8 are used. For better utilization and exploitation of the available abilities of the underlying Big Data cluster, the custom configurations shown in Table 4.3 are used. A feature space consisting of 5 features, namely temperature, humidity, cloud coverage, hour and day is used to build the forecasting models. A dataset of 4 GB size is used for training and testing ML models, where 80% of the input time series dataset are used as a training set and 20% as testing set. For each ML algorithm, the experiment is repeated three times. Afterwards, the mean values are calculated as final performance indicators. Figure 4.6 shows the total time required by the framework to perform the aforementioned task in case of a pre-trained and untrained model pipeline.

In general, the total time T_{total} is strongly related to the complexity of ML models. As this complexity increases, T_{total} required to perform the task will dramatically increases. The base classifier of both RF and GBTs algorithms is the DT algorithm. Consequently, the complexity of RF and GBTs models is higher than the complexity of the DT model. As seen in Figure 4.6, RF and GBTs introduced higher T_{total} than DT and MLR algorithms.

Both GBTs and RF are algorithms for learning ensembles of trees, but the training processes are different. While the GBTs algorithm trains one tree at a time, the RF algorithm can train multiple trees in parallel. This can be seen clearly in Figure 4.6, in which GBTs shows higher T_{total} than RF. In our experiments, both MLR and DT algorithms introduce lower T_{total} compared to RF and GBTs.

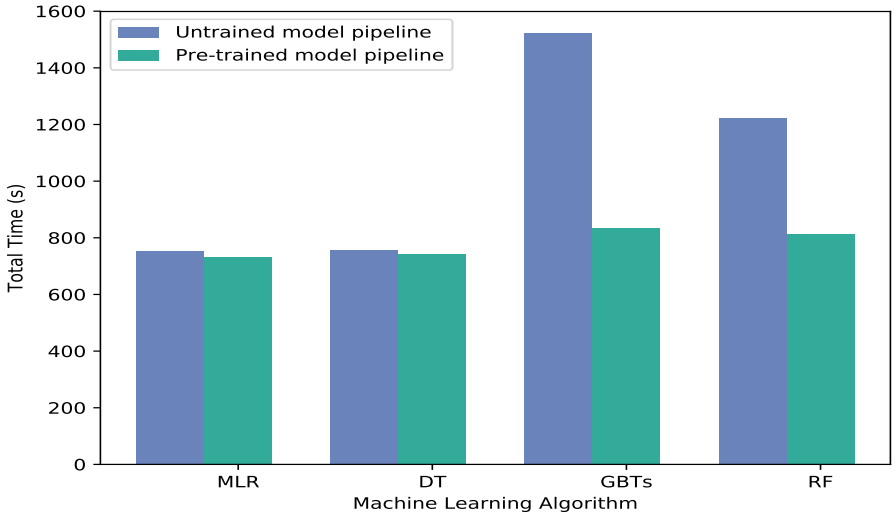


Figure 4.6.: T_{total} required for training and testing models (untrained model pipeline) and for testing (pre-trained model pipeline) on simulated energy multivariate time series dataset with size 4 GB.

The efficiency of storing ML models can clearly be seen in case of complex ML models, namely GBTs and RF models, and will rise with growing complexity of the model. As the complexity of the model increases, the time needed to perform the same task with each new dataset will dramatically increase and the benefit of using pre-trained models will also increase. E.g. by performing forecasting, we gain a time of 690 and 411 seconds in case of GBTs and RF respectively. In contrast to that, only a little time is gained in case of retrieving and reusing simpler models such as MLR and DT as seen in Figure 4.6.

As a result, the recommendation of storing ML models and reusing them in testing (or in production) is higher in case of complex models than for simpler ones. This experimental study gives evidence for the importance of storing and retrieving ML models as a major property in our framework. However, the experiments are performed only with a dataset of 4 GB size. As this size increases, the complexity of the ML models will increase too, paving the road to save and gain more time for performing ML tasks with new datasets based on pre-trained models without the need for training these models.

4.3.2.3. Framework Overhead

In this group of experiments, the framework overhead resulting from performing ML tasks is measured. As a ML task, short-term energy generation forecasting using MLR models is taken into consideration. The algorithm hyperparameter configurations shown in Table 4.8 besides the custom cluster configurations are used. The evaluation instruments the untrained model pipeline, in which the training and the testing steps of ML models are required. The goal of the study is to evaluate the effect of input dataset size on framework performance in terms of the framework overhead defined in Equation (4.2). To achieve that, the size of the input datasets is upscaled to 64 GB, as bigger datasets typically expose more load on the framework infrastructure.

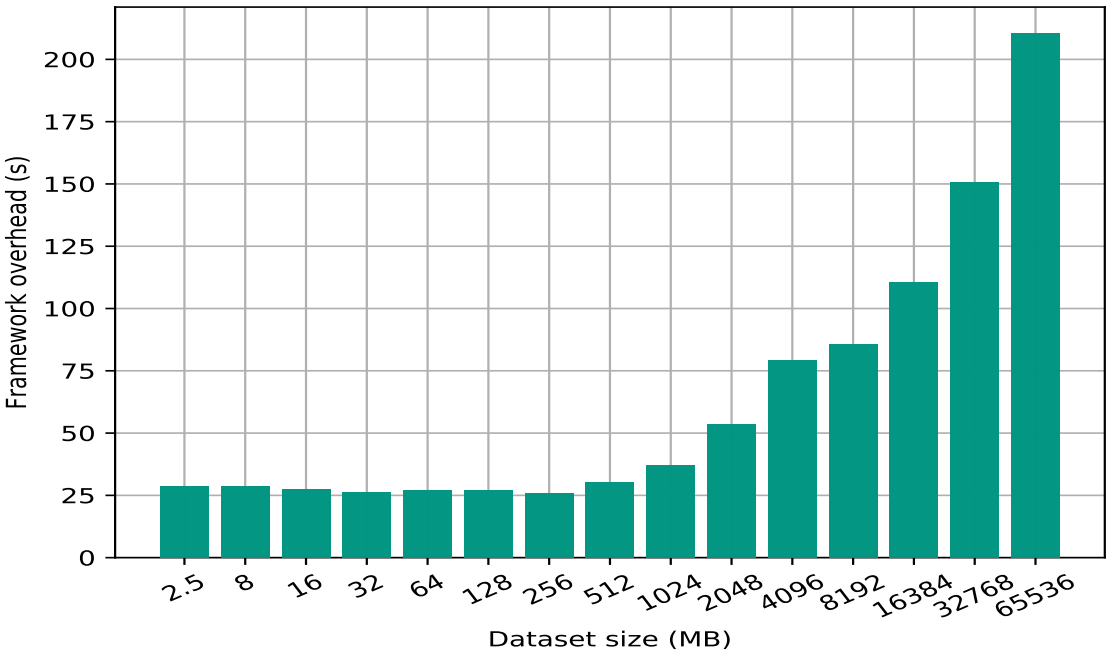


Figure 4.7.: Effect of input datasets size used for training and testing MLR models on the framework overhead.

As defined in Equation (4.2), the framework overhead encompasses communication overhead and database overhead. The obtained results depicted in Figure 4.7 show that the proposed framework introduces an approximately constant communication overhead averaging at around 26 seconds for datasets with sizes up to 512MB. The framework overhead starts to increase for a size of input datasets larger than 512MB. The reason behind this is the additional overhead inside the Big Data environment needed for resource scheduling, coordination and network communications in the cluster. More precisely, an increasing size of the input dataset naturally leads to an increased overhead due to data replication, disk I/O and the serialization of data inside the execution environment of the cluster. A detailed increase in overhead can be seen also in Figure 4.8.

Despite this increment, the introduced framework overhead is still low compared to the execution time spent in performing a ML task as shown in Table 4.10. For example, the portion of framework overhead is 210.47 seconds in the worst case, namely for 65 GB input multivariate time series datasets. Consequently, our evaluation demonstrates that it maintains high performance ML processing with low framework overhead to facilitate and solve ML tasks in Big Data environments.

4.3.2.4. Cluster Utilization Threshold

This section discusses the question “when to use the proposed framework for performing ML tasks more efficiently on a cluster?”. Clearly, the dataset size has a significant effect on the complexity of machine learning models and therefore on runtime performance. As the size of the dataset used for training and testing ML models grows, the complexity of the

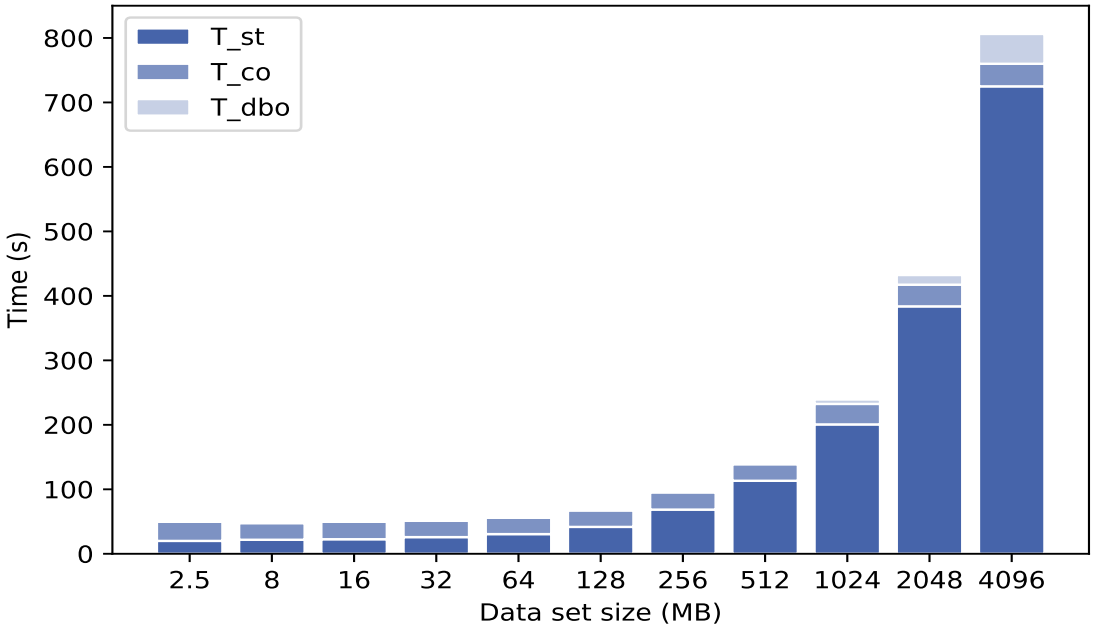


Figure 4.8.: Effect of input datasets size used for training and testing MLR models on the framework overhead (detailed overview).

model increases which dramatically affects the total execution time in our microservice-based framework. While MLR forecasting models have the lower complexity, the GBTs forecasting models represent the models of higher complexity in our evaluation study for the same dataset size as seen in Figure 4.6. The algorithm hyperparameter configurations shown in Table 4.8 are used. We changed the input dataset size between 2.5 MB and 4 GB in the experiments for investigating the effect of dataset size on the framework overhead and execution time. The total time T_{total} is compared to the time required for performing the same task in local and cluster context. The ratio of local time and cluster time is defined as *abs_threshold* in Equation (4.3).

$$abs\_threshold = \frac{T_{local}}{T_{total}} \tag{4.3}$$

where:

- T_{local}: encompasses the total time required to locally execute a machine learning task.

The main idea behind defining *abs_threshold* is to find the dataset size for which the total time in local context exceeds the total time required by the framework to execute tasks in the cluster context. From this point, it is highly recommended to use a cluster. Precisely, to effectively perform machine learning tasks, this ratio should be greater than 1.

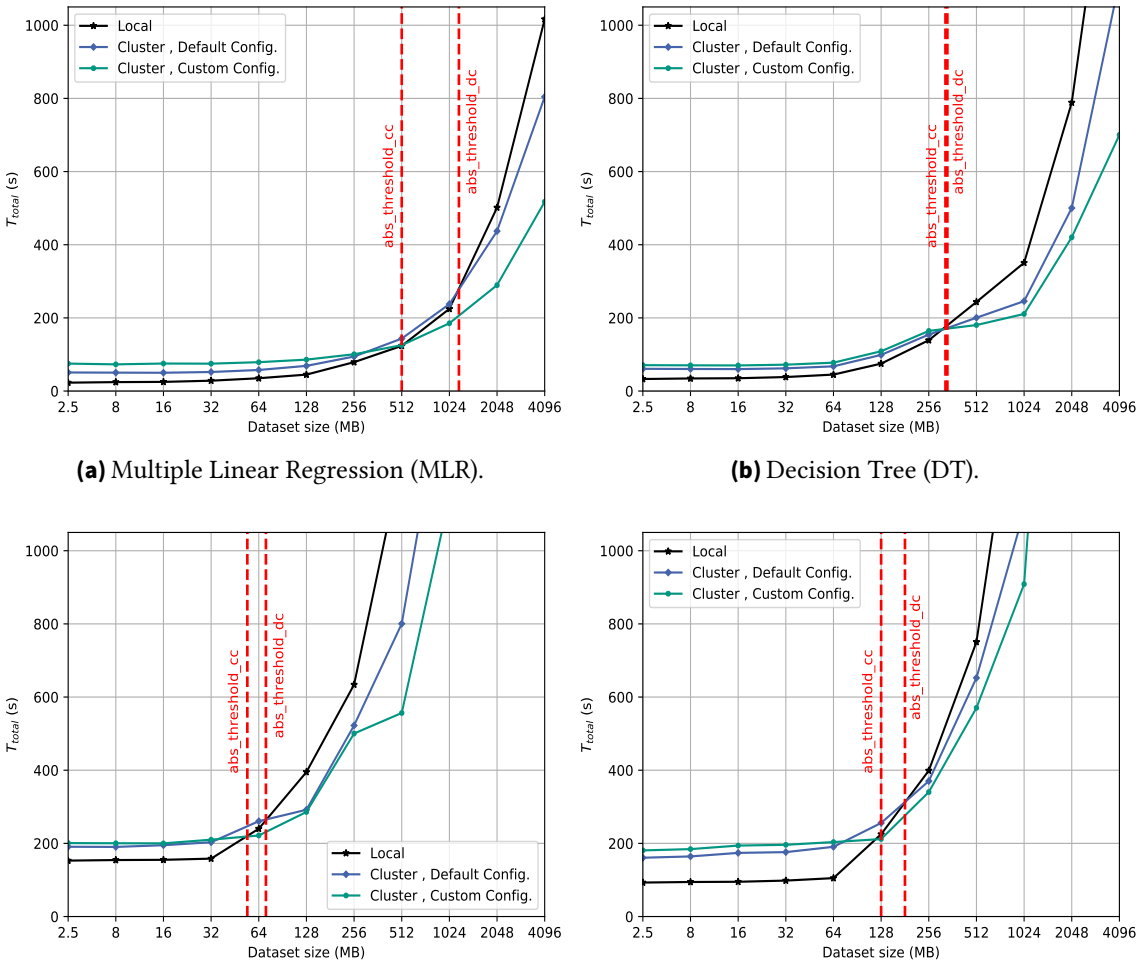
Generally, performing ML tasks in a cluster context introduces additional overhead. The main reason behind this lies in the time cost for resource scheduling, coordination and

Table 4.10.: Execution time and the related overhead required for building MLR models based on different sizes of datasets.

DataSet Size(MB)	Execution Time(s)	Framework Overhead(s)
4	19.99	28.78
8	21.79	28.53
16	22.6	27.41
32	25.88	26.21
64	30.54	27.05
128	41.7	27.13
256	68.54	25.82
512	113.25	30.2
1024	200.66	37.06
2048	383.75	53.54
4096	724.98	79.37
8192	1016.48	85.65
16384	4724.98	110.66
32768	6383.75	150.88
65536	11804.36	210.47

network communications in the cluster. Figure 4.9 shows the mean total time in local and cluster modes, including default and custom configurations, using various dataset sizes. It can be observed that enlarging the dataset size from 2.5 MB to 64 GB has no significant effect on both T_{local} and T_{total}.

As seen in Figure 4.9a and for data less or equal to 256 MB, T_{total} in both cluster modes is greater than T_{local} in local mode which can be explained by the added overhead for processing the application on the cluster. Thus, running Spark applications locally for these dataset sizes is more efficient. For a dataset size less than 256 MB, T_{total} with custom configurations is greater than T_{total} with default configurations, since two additional nodes are used in these configurations where each of them introduces an overhead. As expected, when the dataset size grows larger, utilizing a cluster becomes more desirable which is shown by the intersection points highlighted by the two red lines, where these points depend on the configurations. As the *abs_threshold_cc* (cc: custom configurations) is found at a dataset size of 512 MB making the custom configurations the most efficient beyond that point, the *abs_threshold_dc* (dc: default configurations) lies at a dataset size of >= 1024 MB. Consequently, the computing power of the cluster can be seen and the



(a) Multiple Linear Regression (MLR).

(b) Decision Tree (DT).

(c) Gradient-boosted trees (GBTs).

(d) Random Forest (RF).

Figure 4.9.: Mean T_{total} in case of local and cluster (default, custom) configurations mode to determine the *abs_threshold* for MLR, DT, RF and GBTs algorithms.

time consumed locally to perform a task will exceed the time required to perform the same task on cluster. Therefore, it is recommended here to use the cluster.

Comparing Figures 4.9a, 4.9b, 4.9c and 4.9d, we conducted that as the complexity of ML models increases, the *abs_threshold* is early arrived. The reason is that the complex models need more calculation costs. As a result, the performance in the cluster context will earlier outperform the performance in the local context because of the power of the underlying deployed Big Data cluster. Concerning the GBTs model which represents the most complex model in our benchmark evaluation, the *abs_threshold* is arrived at an input dataset in size of about 64 MB. In contrast to that, lower complexity models such as DT models introduced *abs_threshold* for 300 MB.

As mentioned before, there is an inherent overhead in the framework arising from e.g. database communication and the use of the cluster. The smaller this overhead is compared to Spark's execution time, the more efficient the framework is. To gain insight into how

the efficiency of the framework varies as the dataset grows, a new threshold, referred as *min_threshold*, is defined and formulated in Equation (4.4):

$$min\_threshold = \frac{T_{exe}}{T_{fo}} \tag{4.4}$$

where:

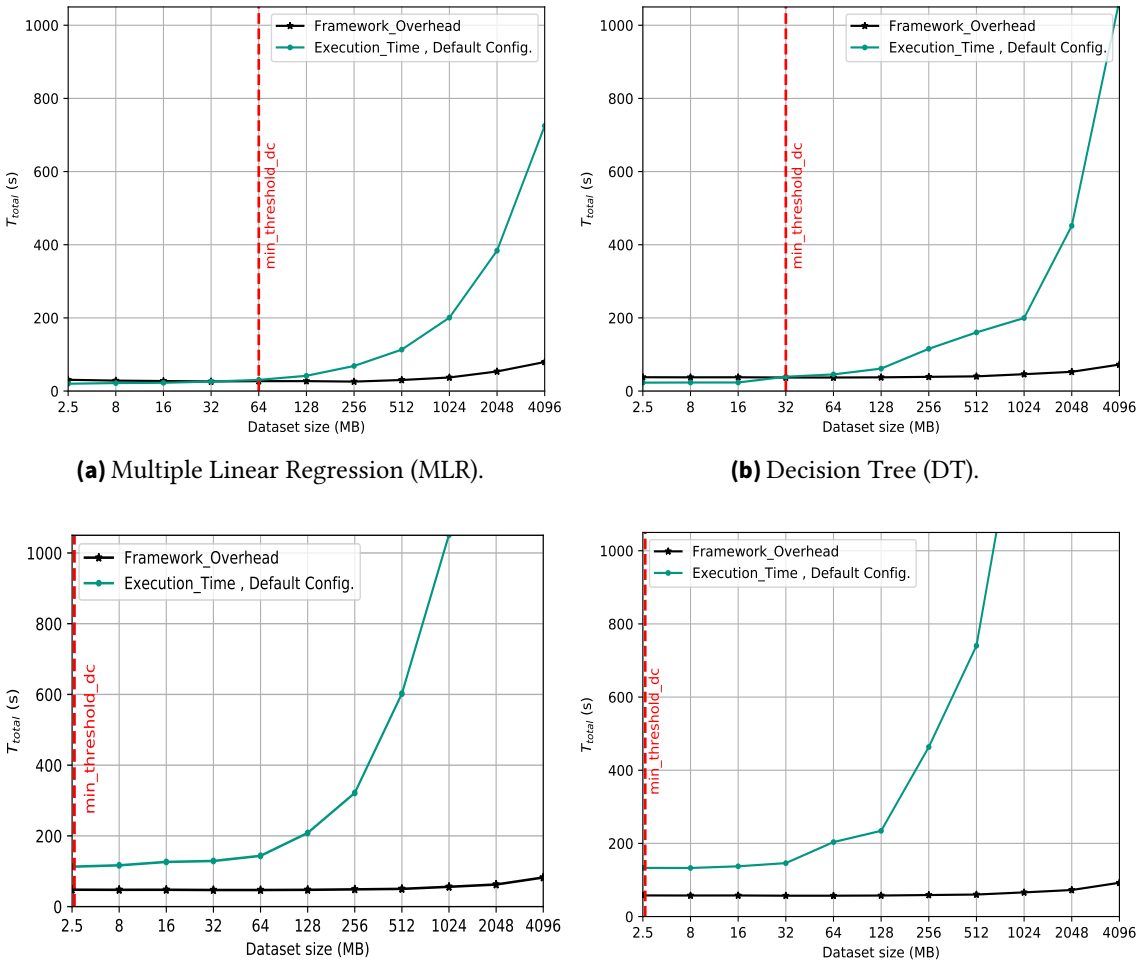
- T_{exe} is the execution time required by Apache Spark to perform a ML task in context of pre-trained pipelines or untrained pipelines, same as T_{st} defined before.

This threshold is defined based on the fact that for an efficient execution of a task, the overhead time should not exceed the time required for the execution. Consequently, to effectively perform machine learning tasks, this threshold should be greater than 1. The main difference between *min_threshold* and *abs_threshold* lies in the context, in which they are calculated. While *abs_threshold* compares the total time required to perform a machine learning task in local and cluster context, the other one is calculated only in cluster context comparing the framework overhead with the execution for different dataset size. As a result, we discovered the point at which it is recommended to use our framework in cluster context.

This group of experiments is conducted using the default cluster configurations summarized in Table 4.3 and also repeated three times for more robust results. The obtained mean results, presented in Figure 4.10 show that *min_threshold* has the same behavior of *abs_threshold*. It is evident that for very small dataset sizes, the *min_threshold* is less than 1 since more time is spent on T_{fo} than T_{exe}. The *min_threshold* comes closest to 1 at the size of 64 MB and 32 MB for MLR and DT respectively as seen in Figures 4.10a and 4.10b. Beyond this point, T_{exe} starts to exceed T_{fo} which implies that for larger dataset sizes it is recommended to use the framework in cluster context. Precisely, the gap between T_{fo} and T_{exe} increases proportionally to the dataset size, since T_{exe} is strongly dependent on it. As the complexity of the model increases, the *min_threshold* is shifted to meet smaller dataset sizes i.e., 2.5 MB as seen in Figures 4.10c and 4.10d. Combining the results of *abs_threshold* and *min_threshold*, it is recommended to perform ML tasks using the proposed microservice-based framework if both of these thresholds are greater than 1.

4.4. Summary

In this chapter, we presented a new highly scalable generic microservice-based architecture, our solution to support 2A non-expert users to perform ML tasks in Big Data environments. We started in Section 4.1 by introducing the problem statement, in which the challenges and the problem to be solved are explained. In Section 4.2, the conceptual microservice-based architecture is introduced including the different layers that communicate with each other to achieve the main goals of the framework. We explained the different layers,



(a) Multiple Linear Regression (MLR).

(b) Decision Tree (DT).

(c) Gradient-boosted trees (GBTs).

(d) Random Forest (RF).

Figure 4.10.: Mean T_{total} in case of local and cluster (default, custom) configurations mode to determine the *min_threshold* for MLR, DT, RF and GBTs algorithms.

their services and the REST-APIs developed in this thesis. To understand the interaction between services, a brief execution workflow was introduced.

We evaluated our solution in Section 4.3. We started by presenting the experimental setup and configurations required to conduct our experiments. In a comprehensive evaluation study, the advantage of storing and retrieving ML models is demonstrated. The results also show that the caching of RDDs in Apache Spark plays an essential role in saving the execution time required for performing the task on the cluster. Precisely, the training and prediction time is reduced by approximately 75% when combining lazy evaluation with caching techniques.

Moreover, by measuring the framework overhead and comparing it to the model calculation time, it could be demonstrated that the proposed framework introduces an acceptable low overhead relative to an increasing size of an input dataset. The advantage of storing and retrieving ML models as a major property of our solution is also investigated. We have seen

that it is of great importance to store ML models in order to be used later in production for other tasks, especially in the case of complex models. For efficient utilization of the proposed framework, certain thresholds are defined to determine the dataset size, in which it is highly recommended to use clusters in favor of single computers for performing a given ML task.

As a result from the evaluation, our framework introduced a suitable solution to enhance the applicability of the trial-and-error approach for building ML models in cluster environments. Such suitability is derived from different reasons. Firstly, hiding the low level complexities of big data infrastructure from the user. Secondly, the low overhead during performing ML tasks. Thirdly, the advantage of storing and retrieving ML models. Fourthly, caching the input energy time series datasets which positively affects the execution performance of the framework. Lastly, the best practice for using such framework for performing ML tasks on clusters.

5. Characterizing Energy Time Series Datasets

As established in the motivation, the accurate forecasting of future energy demand and renewable energy generation provides better sustainability and intelligent decision making in managing energy resources at grid level. A plethora of machine learning algorithms are existing in the literature to forecast energy time series datasets. In order to select the respective algorithm for a given time series dataset, the end user often uses a trial-and-error approach which takes a long time for constructing multiple models. To address this issue, meta learning is existing attempting to find the most appropriate model for forecasting based on the characteristics of the given datasets [81][151][25].

The characteristics of energy time series datasets may provide useful insights into which methods are most appropriate for forecasting [55][16]. To this end, we propose Descriptive Statistics Time based Meta Features (DSTMF) as a new group of meta features to describe energy time series datasets. Descriptive statistics are a set of summary statistic properties that quantitatively describes the numerical properties of a dataset and summarizes its features with regard to the distribution of data. The well known five-number summary is a set of descriptive statistics that characterizes the dataset in five sample percentiles, namely minimum, maximum, median, 1st Quartile and 2nd Quartile which are derived from the well-known boxplot/box-and-whisker plot [79].

In this chapter, we present our meta learning approach including a new type of meta features for selecting the most appropriate forecasting model for energy time series datasets. We start in Section 5.1 by explaining the problem statement including the main challenges, goals and problems we aim to solve and the research question we answer in this chapter. Then, Descriptive Statistics Time-based Meta Features (DSTMF) as a new art of meta features is presented in Section 5.2.1. For efficient instrumenting of meta features, we propose Energy Meta Learning System (EMLS) and Encoded Energy Meta Learning System

Parts of this chapter are reproduced from:

- S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Descriptive Statistics Time-based Meta Features (DSTMF) Constructing a Better Set of Meta Features for Model Selection in Energy Time Series Forecasting". In: Proceedings of the 3rd International Conference on Applications of Intelligent Systems, pp. 1–6. doi: 10.1145/3378184.3378221.
- S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Incorporating Unsupervised Deep Learning into Meta Learning for Energy Time Series Forecasting". In: Proceedings of the Future Technologies Conference, pp. 326–345. doi: 0.1007/978-3-030-63128-4_25.

(EEMLS) in Sections 5.2.2 and 5.2.3. respectively. While the original representation of meta features is used in the first one, we encode the meta features using autoencoders in the second system to leverage the advantage of unsupervised deep learning in enhancing the predictive performance of meta learning classification models.

We evaluate our work in Section 5.3 in three points of view. In the first one, similarity-based clustering analysis is performed to measure the accuracy of the extracted meta features in characterizing energy time series datasets. In the second and third one, we evaluate this accuracy by incorporating the original and encoded representation of meta features into a complete meta learning scenario for energy time series model selection. The research contributions presented in this chapter were the main topics of our papers in [143][144].

5.1. Problem Statement

In view of more renewable energy being injected in the energy systems, accurate forecasting of electrical load and generation has attracted significant attention for optimizing the usage of energy to achieve better management as well as distribution of energy at grid level. To this end, there has been historically a lot of research around performing energy load and generation forecasting, in which the users often used the well-known trial-and-error approach for finding the best forecasting model for a given time series. Based on the utilized modeling approach, we can classify these forecasting models into five categories, namely artificial intelligence models [43][179][97], averaging models [170], time series models [11], regression models [12][69][84], as well as hybrid models [190].

The trial-and-error approach is based on the idea that all possible combinations of the learning algorithms with their relevant hyperparameters are tried for each task to find the most suitable one. However, this is typically tedious. It wastes the resources for constructing multiple models which can take a long time especially in the case of large datasets to be forecasted. Moreover, expert knowledge is required to perform such tasks. To address this issue, some meta learning approaches have been proposed for automatically finding the most appropriate model for time series forecasting based on the characteristics of input time series datasets [92][6][151][25]. These approaches consist of three main input spaces, namely problems, algorithms and meta feature space. The problem space is reflected by a datasets pool that includes a variety of datasets for training models and extracting meta features. The algorithm space includes a set of forecasting algorithms, from which the most suitable candidate is selected as the best solution for forecasting. The meta feature space contains a set of meta features that capture the characteristics of datasets.

A meta learning approach is based on the fact that the good feature set capturing the characteristics of a time series dataset gives powerful insights into the future behavior of this dataset paving the road for determining the most appropriate forecasting algorithm. According to that, the better the used meta features characterizes the behavior of a time series dataset, the more accurate the selection of the best forecasting algorithm will be.

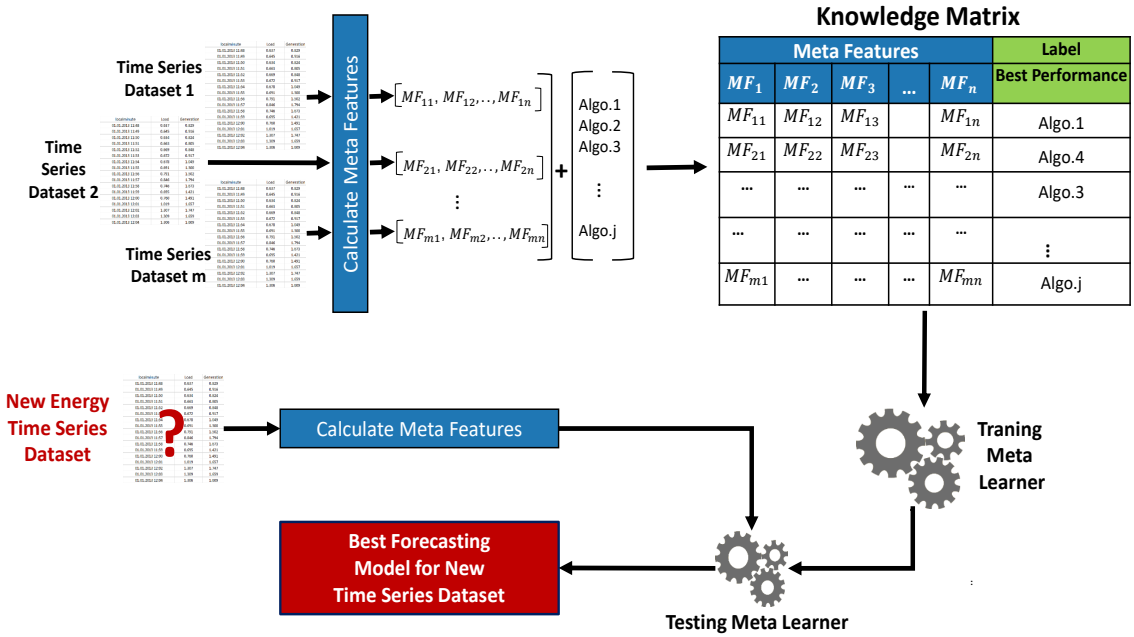


Figure 5.1.: General methodology of meta learning for time series model selection [143].

Typically, a meta learning approach solves the problem of model selection by formulating it as a supervised learning task incorporating training and testing phases. The main task of the training is that an algorithm, referred to as meta learner, learns the mapping between available forecasting algorithms and the corresponding set of meta data indicator values for the same dataset. Therefore, a set of meta examples are needed, that correspond to the already mentioned set of learning datasets which are accordingly tagged by predictors and labels as shown in Figure 5.1. The predictors correspond to the meta features extracted to describe the datasets. The labels indicate the most appropriate algorithms. In the testing phase, given a new time series to be forecasted, the meta features are extracted. Based on these features, the trained meta learner will suggest the most appropriate model.

Building on that, the good characteristics of energy time series datasets plays an essential role in precisely achieving the main goal of meta learning and assigning the best model for the input time series datasets. Such characteristics are considered as a challenge in building an efficient meta learning model. In this chapter, we tackle this challenge and answer the research question RQ2 “How to efficiently characterize energy time series datasets to enhance the performance of automated model selection?” by proposing Descriptive Statistics Time-based Meta Features (DSTMF) as a new art of meta feature to capture the deep characteristics of energy time series datasets.

5.2. Proposed Solution

To address the problem stated in Section 5.1, we propose Descriptive Statistics Time based Meta Features (DSTMF) as a new group of meta features to describe energy time series

datasets. Furthermore, we propose the Energy Meta Learning System (EMLS) and Encoded Energy Meta Learning System (EEMLS) to efficiently instrument the new art of meta features in a meta learning scenario for recommending the most adequate forecasting model.

5.2.1. Descriptive Statistics Time-Based Meta Features (DSTMF)

In this section, we introduce Descriptive Statistics Time-based Meta Features (DSTMF), our new type of meta feature to characterize energy time series datasets. As its name implies, DSTMF is based on the descriptive statistics which are a set of summary statistic properties that quantitatively describe the numerical properties of a dataset and summarize its features with regard to the distribution of data. The well known five-number summary is a set of descriptive statistics that characterizes the dataset in five sample percentiles, namely minimum, maximum, median, 1st Quartile and 3rd Quartile which are derived from the well-known boxplot/box-and-whisker plot.

Five-number summary statistics provide information about the location (from the median), the spread (from the quartiles) and the range (from the minimum and maximum of the observations). While the main advantage of the five-number summary introduced with boxplots is to detect outliers in datasets, we leverage it to support our meta learner with some new meta features capturing the behavior of time series datasets. The proposed DSTMF approach is based on extracting the aforementioned five-number summary statistics extended by the arithmetic mean for different aggregation levels of the input datasets as seen in Figure 5.2. For each input time series dataset in the problem space, we have n possible aggregation levels. In this context, we distinguish between calculating DSTMF in terms of load and generation time series datasets.

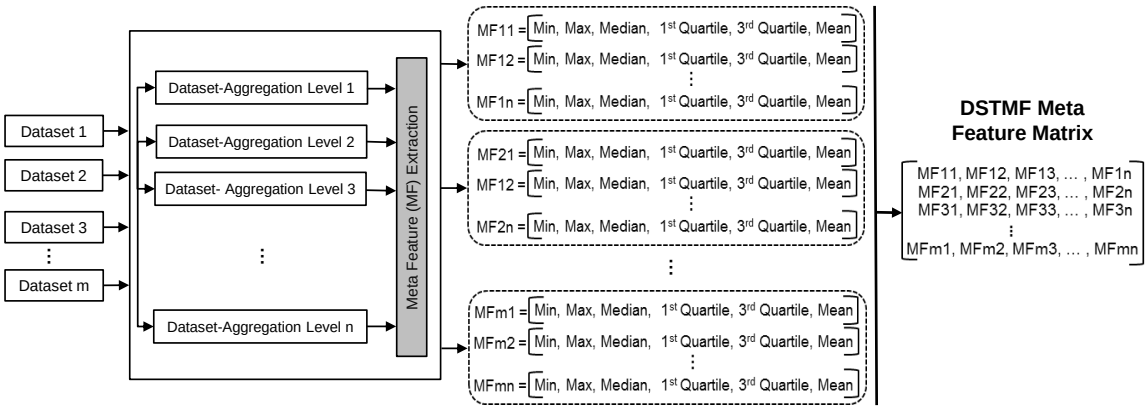


Figure 5.2.: Methodology of extracting Descriptive Statistics Time-based Meta Features (DSTMF).

In the case of load time series datasets, 10 target aggregation levels are defined, namely 1-hour, daily, monthly, weekend, workday, workday_sum, winter, summer, spring and autumn. The final meta feature set results from combining the meta features of the aggregations with that from the original dataset. It gives a detailed representation and

characterization of the input time series at different aggregation levels. The reason for this approach is that, if we have an input time series with 1-hour measurement values, capturing the characteristics of a time series dataset only at the fine-grained aggregation level is not always a good idea, because of the effect of changing in the readings for the same time point from day to day. E.g., assume that we have a time series of an energy load for a building. On one day, the peak energy load in the morning is at 7 an (work-day) while on the next day the peak is at 9 an (weekend). Such changes in the energy load may affect the accuracy of understanding the time series and capturing the deep characteristics of it. As a result, if we aggregate the hourly energy load in a daily energy load moving toward a coarse-grained energy load, we can hide these changes and reduce the bias toward specific values at the fine-grained aggregation level.

In case of power generation time series datasets, some of the aforementioned aggregation levels do not exist, e.g. workdays and weekends. The reason for that lies in the fact that the power generation of photovoltage is mainly dependent on the weather and not affected by human behaviour. As a result, such aggregation levels are excluded and DSTMF is calculated for the rest of the resulting aggregation levels of the generation time series datasets.

To operationalize DSTMF, we start with the original input time series dataset extracting the aforementioned meta features. Then, the dataset is aggregated to each target aggregation level larger than the original aggregation level. An aggregation level A is considered to be larger than an aggregation level B, if the time horizon between the data points in A is larger than the time horizon between data points in B. For instance, given an input time series dataset with daily aggregation level. From this, a load time series dataset with 1-hour aggregation level cannot be derived without losing the accuracy of respective time series measurements. Therefore, the possible aggregation levels will be monthly, weekend, workday, workday_sum, winter, summer, spring and autumn.

For each possible aggregation level, the five-number summary in addition to the arithmetic mean is calculated. The results are combined to one meta feature set forming the first row in the matrix illustrated in Figure 5.2. We repeat this process for all input time series datasets to have the final shape of the meta features dataset as a matrix, as shown in Figure 5.2 which will be used by meta learners to select the most appropriate forecasting model. While each row in this matrix corresponds to a specific input time series dataset from 1 to m, each column is calculated for a specific possible aggregation level from 1 to n. In order to avoid having missing values in the final meta features dataset, one has to consider one important property that all input time series datasets should have, i.e., the same granularity level. This ensures that we have the same number of possible aggregation levels producing the same number of meta features for each input time series dataset.

5.2.2. Energy Meta Learning System (EMLS)

As we have mentioned before, to assess the effectiveness of DSTMF in characterizing energy time series datasets, EMLS as depicted in phase 1 and phase 2 of Algorithm 1 is

conceptualized and implemented. Our system will support the selection of the appropriate forecasting algorithm for the respective dataset. A variety of meta features groups are extracted in the present work, namely simple, statistical, information-theoretic, time series and DSTMF meta features. Both Random Forecast (RF) and Artificial Neural Network (ANN) are involved in EMLS as the meta learners. To ensure better applicability of the classification rules by the meta learner, an efficient feature selection and importance analysis procedure is integrated into EMLS.

Algorithm 1 Energy Meta Learning System (EMLS)

Phase 1: Prepare energy time series datasets and build meta learning classification model

Input:

$X = \{x_1, x_2, \ldots, x_n\}$: Problem space (energy time series datasets)
$A = \{a_1, a_2, \ldots, a_m\}$: Algorithm space
$Agg = \{Daily, Weekly, Monthly, ...\}$: Aggregation levels of X
PP: Set of functions to preprocess X
MF: Set of functions to extract meta features categories
$DSTMF$: Set of functions to extract DSTMF meta features
FF: Set of functions to calculate forecasting features

Output:

Meta learning classification model

preprocessing and features extraction of problem space X
for i = 1 to n in X **do**
    Apply PP to preprocess x_i
for i = 1 to n in X **do**
    Calculate forecasting features FF for x_i
Extract Meta knowledge
for i = 1 to n in X **do**
    Split x_i into training set $x_i\_training$ and test set $x_i\_testing$
    for j = 1 to n in Agg **do**
        Aggregate $x_i\_training$ based on Agg
        Calculate meta features $DSTMF$ on the aggregated datasets
    Calculate meta features MF on training set $x_i\_training$
    for k = 1 to m in A **do**
        Train model a_k on $x_i\_training$
        Test model a_k on $x_i\_testing$ and save forecasting error
    Select model with best performance measure as a label for x_i
Build Knowledge Matrix (KM)
Split KM into $KM\_training$ and $KM\_testing$
train and test meta learning classification model on $KM\_training$ and $KM\_testing$

Algorithm 1 - Phase 2: Model selection for a new energy time series dataset

Input:

Meta learning classification model
x_{new}: energy time series dataset to classify (a new building)
FF and $DSTMF$ functions

Output:

The adequate forecasting model

18: Apply PP to preprocess time series dataset x_{new}
19: Aggregate x_{new} based on Agg
20: Apply $DSTMF$ functions on the aggregated results
21: Apply MF functions on x_{new}
22: Pass the extracted meta features of x_{new} to the meta learning classification model to predict the best forecasting model for x_{new}
23: Update KM with x_{new}

One of the main advantages of EMLS proposed in this thesis lies in the dispensing with the usage of physical features to describe the buildings. EMLS indirectly and anonymously captures physical and other social properties influencing the load behavior of a building, which are usually difficult to get explicitly due to privacy and security issues. Such information is implicitly contained with time series load data of the building and can be efficiently instrumented as meta features. Moreover, EMLS presents a generic meta learning system which is able to handle a variety of energy use cases, not only for buildings but also for other energy time series use cases. Regardless of the concrete goal of forecasting, EMLS is able to assign an adequate model to a given energy time series dataset. Precisely, EMLS could be utilized in the case of short, mid and long term for both energy load and generation forecasting.

EMLS is explained utilizing the Energo+ time series dataset presented in 2.4.1 with loads for 200 buildings. As depicted in Algorithm 1, EMLS incorporates two main phases, in which the meta learning model is first trained on a set of time series datasets describing the electricity loads of buildings and afterwards tested on new load datasets to recommend the most appropriate short term forecasting model for them. In the following, the approach outlined in Algorithm 1 is briefly explained.

Phase 1: Prepare Energy Time Series Datasets and Build Meta Learning Classification Model

1. **Time series preprocessing.** To ensure a better quality of time series datasets which leads to an accurate load forecasting, EMLS applies a preprocessing procedure. First, the negative values are automatically set equal to zero. After that, Hampel Filter [112] which is a widely used method to detect outliers in time series is used. After performing normalization in which time series are re-scaled to values between zero and one, linear interpolation defined in [52] is used to fill gaps if they are small, ranged between one missing value and up to eight missing values (2 hours).

To fill bigger gaps, a more elaborate method is required. As we often have a high similarity of the same days every week, it is possible to use values from past or future weeks to impute the missing data with. To compensate for some of the variance, we impute the mean of several weeks taken from the past and the future of the currently considered gap of missing data. These samples are only used if they themselves don't contain any missing data.

2. **Short-term Load Forecasting Scenario.** Depending on the use-case, different forecasting horizons can be considered. For example, larger forecasting horizons are considered for maintenance related issues. Most studies however focus on short-term or very-short term forecasting, because they are the most important in the areas of storage control, automatic generation control, unit commitment and the electricity market [169]. In fact, most energy is traded in one-day-ahead markets, which raises the significance of one-day-ahead forecasts [14]. For this reason, we will focus on the task of predicting a solar power generation time series one-day-ahead in the form of multi-step forecasts.

   The algorithm space is comprised of all the algorithms in the target space of the meta learning multi-class classification problem. In other words, it contains all algorithms which can be recommended by the meta learning system. For each input time series dataset, we build forecasting models including 4 algorithms developed for time series forecasting, namely DT, RF, GBTs and LR. Further details about the way in which our input time series datasets are forecasted using these algorithms, is provided in Section 5.3.1.

3. **Meta Features Extraction.** As mentioned before, the characteristics of energy time series datasets paves the road into which methods are most appropriate for forecasting. As seen in Algorithm 1, the next step in the proposed EMLS is to extract the meta information required to describe time series datasets. Such meta features are extracted on the training parts of time series datasets. According to the DSTMF methodology depicted in Figure 5.2, the original input time series datasets which have a time resolution of 15 minutes, are then aggregated to 1-hour, daily, monthly, weekend, weekend sum, workday, workday sum, winter, summer, spring and autumn. For each aggregated dataset, the five-number summary and the arithmetic mean are calculated resulting in six meta features.

   For robust evaluation, it is of great importance to compare DSTMF to state-of-the-art meta features existing in literature in the field of energy. (1) Simple meta features (S.F.) can be directly derived from time series datasets, e.g. the number of samples and the number of attributes. (2) Statistical meta features (St.F.) describe the statistical properties of time series datasets such as Standard Deviation (SD), skewness, average, variance, median, kurtosis, step, to name a few. (3) Information-theoretic meta features (I.F.) are directly based on entropy measures, e.g. entropy and noise-signal ratio [29]. (4) Time series meta features (T.F.) are types of statistical features but are grouped in another category because they are strongly related to time series datasets [48], e.g. lumpiness, stability, max level shift, max var shift, max kl shift, crossing

points, flat spots, stl features, acf features, pacf features and heterogeneity, to name a few. All these meta features could be computed using methods available in R [157].

The following matrix sketches a formal representation of the extracted meta features, whereby each row corresponds to a specific time series dataset:

$$\begin{pmatrix} S.F., ..., St.F., ..., I.F., ..., T.F., ..., DSTMF, ... \\ S.F., ..., St.F., ..., I.F., ..., T.F., ..., DSTMF, ... \\ ... \\ S.F., ..., St.F., ..., I.F., ..., T.F., ..., DSTMF, ... \end{pmatrix}$$

The DSTMF meta features group is constructed by combining the calculated descriptive statistics from all aggregation levels and the original time series. With six values per aggregation level, there are a total of 72 meta features. To discover the effectiveness of DSTMF, seven different groups of meta features are proposed, namely simple, statistical, information-theoretic, time series, DSTMF, all meta features without DSTMF and all meta features including DSTMF. The distribution of meta features is 2, 7, 2, 45, 72, 56 and 128 meta features for the aforementioned groups respectively. After extracting the meta features, EMLS will construct the meta knowledge in the form of a knowledge matrix. In this matrix, each row corresponds to one building containing meta features of this building as predictors and the best forecasting model as a label. This step is followed by training random forest and artificial neural networks as meta learners to learn the mapping between the extracted meta features and the best forecasting models. Based on this mapping, the resulting meta learning classification model is used later to recommend the best forecasting model for new buildings as depicted in Algorithm 1. The results of training and testing meta learners are explained and discussed in more detail in Section 5.3.

Phase 2: Model Selection for New Energy Time Series Datasets

In this phase, the best short term load forecasting model for new buildings is recommended. According to Algorithm 1, we start by preprocessing time series and aggregating them in different aggregation levels. After that, DSTMF and the other groups of meta features are extracted. These meta features will then be passed to the meta learning classification model setup and trained in the previous phase to assign the most appropriate short term load forecasting model for them. After the new load datasets of buildings have been classified, the knowledge matrix is updated for improving the predictive performance of the meta learner over the time.

5.2.3. Encoded Energy Meta Learning System (EEMLS)

For better exploitation of meta information included in the extracted meta features, this section discusses the use of deep learning to encode the extracted meta features by autoencoders. The main idea behind autoencoders is that we use unsupervised learning for leveraging the advantage of neural networks for the task of meta features representation

learning. We design a neural network architecture such that we impose a bottleneck in the network forcing a compressed knowledge representation of meta features.

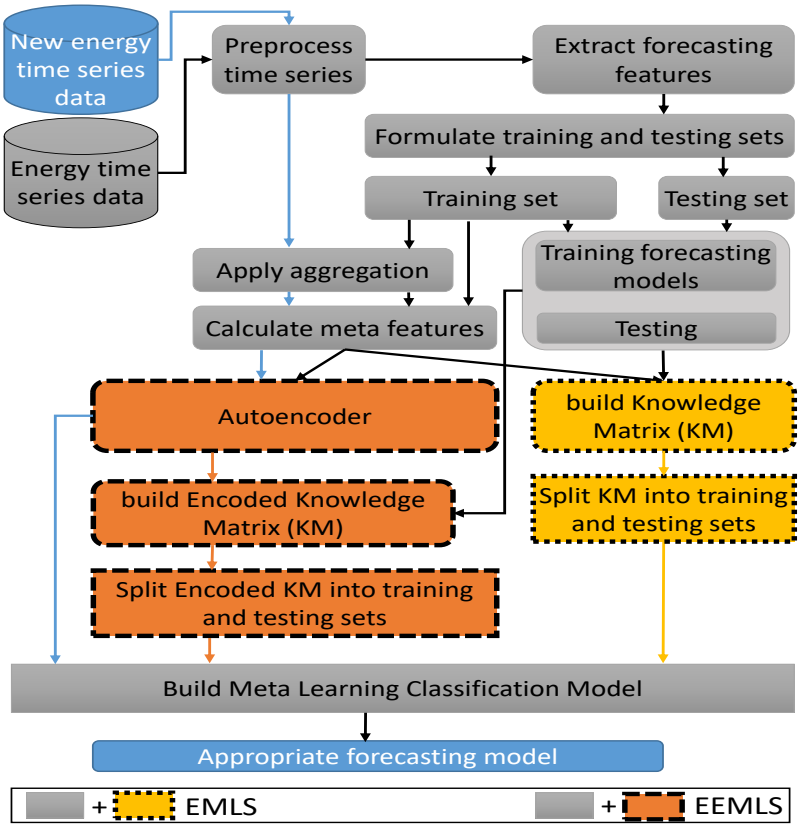


Figure 5.3.: Methodology of EMLS and EEMLS [144].

Figure [**EEMLS**] demonstrates the main methodological approach of EEMLS. As seen is this figure, EEMLS follows the same methodology of EMLS with the main difference that it utilizes autoencoders to encode the input meta features in a new representation form for capturing the deep characteristics of energy time series datasets in an "Encoded Knowledge Matrix". We adapted EMLS to employ autoencoders by adding an encoding step between the steps 15 and 16 in the aforementioned Algorithm 1.

5.3. Evaluation

The key focus of the evaluation is to investigate the accuracy of the new art of meta features, DSTMF, in characterizing the energy time series datasets. We aim to support meta learner with new art of meta feature to enhance its performance in recommending the best forecasting model from the ones presented in 5.3.1. To this end, we conducted three groups of experiment on the dataset presented in 2.4.1. Based on a similarity-based clustering analysis in the first group of experiments, the potential of DSTMF's meta features for capturing deep characteristics of energy load time series datasets is evaluated and compared

to other state-of-the-art meta features. In the second and third group of experiments, we build a complete meta learning use case according to the main methodologies of EMLS and EEMLS to evaluate DSTMF in an original and encoded representation form. In these experiments, we compare the predictive performance of our meta learning classification model using DSTMF with the predictive performance using other state-of-the-art meta features introduced in the literature. The encoded representation of meta features enabled us to leverage the advantage of unsupervised deep learning in the context of energy time series model selection as will be seen in the following sections.

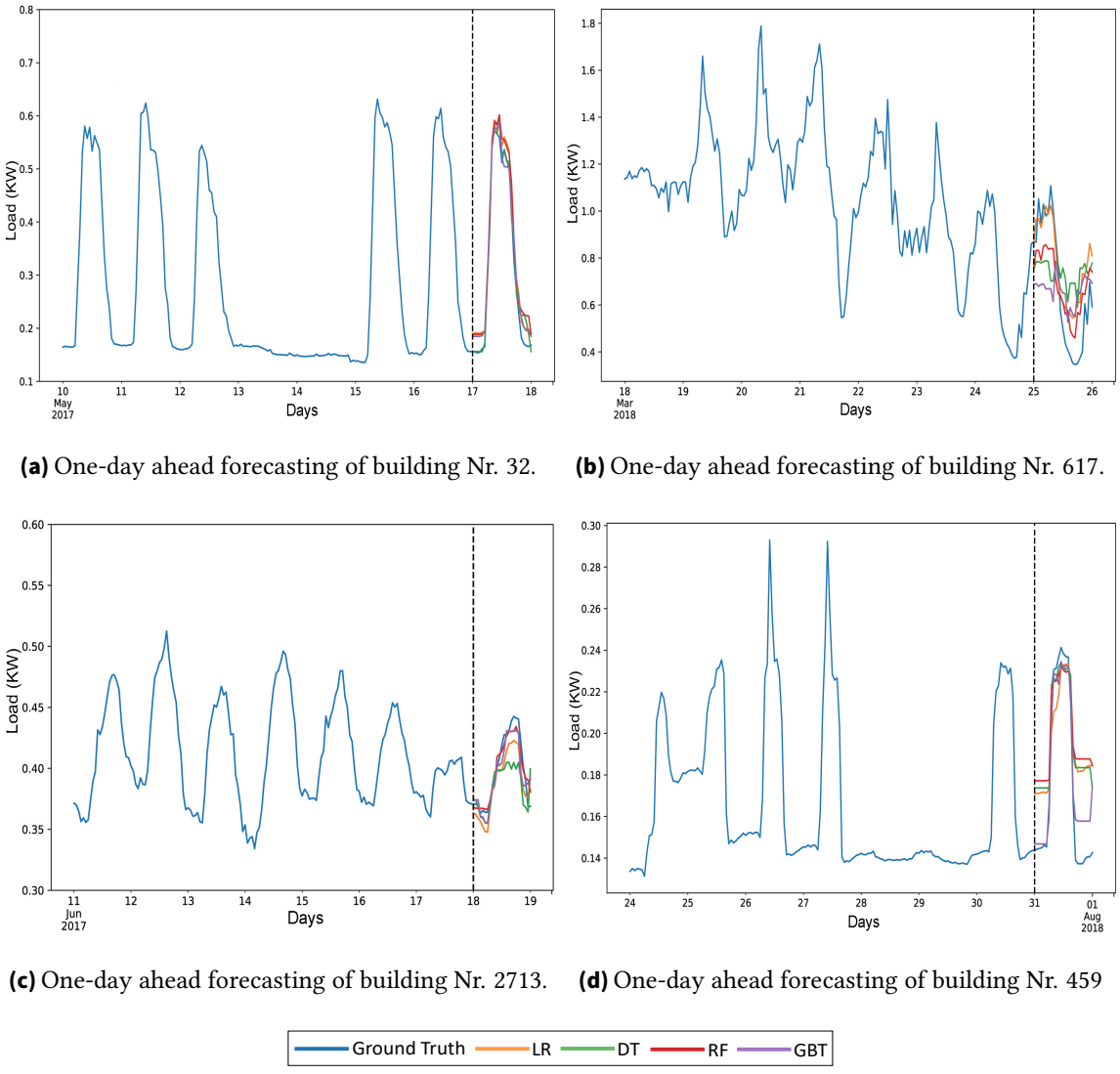
5.3.1. Use Case Study: Short-Term Load Forecasting Scenario

Accurate load forecasting of individual buildings has attracted significant attention in recent years, especially in energy management systems which monitor, optimize and control the smart grid energy market [97]. The reason for that lies in the fact that an accurate knowledge about the future energy patterns helps to reshape the load, paving the road for better management as well as distribution of energy at grid level. However, there are many approaches to predict the electrical load. In the past, physical models were often used leveraging the physical knowledge of buildings and domain expertise to forecast electrical load [86][89][99]. The emergence of automatic data collection and other technological advancements enabled the use of data-driven models [52][9].

Massive amounts of data can be used to unfold the underlying knowledge without expert knowledge about the domain. In addition to the raw load data, exogenous variables like weather or calendar data can be used to support the forecasting model which can be categorized by their forecasting horizon as: Short Term Load Forecasting (STLF) with a horizon up to one week is important for real-time operations and short-term decisions. Medium Term- (MTLF) and Long Term Load Forecasting (LTLF) are used for monthly or yearly planning and energy management.

In this chapter, the focus lies on one-day ahead STLF data driven models with few calendar-based exogenous variables. The reason for choosing such a use case lies in the increasing importance of energy load forecasting in the modern power systems [97][184][54][8][77] In our evaluation, each time series dataset is a series of load (kW) values with sampling rate of 15 minutes which leads to 96 values to be predicted for one day. Based on the fact that the periodicity is one of the main characteristics in electricity load time series datasets, calendar features such as hour, day, month and weekend are extracted for each building as exogenous variables to be used as predictors for short-term forecasting. Additionally, we choose to use the load values of one-day-ago and the corresponding value in one-week-ago as forecasting features, as the energy load patterns of the investigated time series of office buildings likely have daily and weekly seasonality.

DT, RF, GBTs and LR are utilized to forecast one day ahead load. Generally and for all buildings, only the last year is used for testing and the rest for training. Based on the main idea of the meta learning approach, the model with best forecasting performance is considered as a label in the final knowledge matrix to represent the best solution for



(a) One-day ahead forecasting of building Nr. 32.

(b) One-day ahead forecasting of building Nr. 617.

(c) One-day ahead forecasting of building Nr. 2713.

(d) One-day ahead forecasting of building Nr. 459

Figure 5.4.: Predictive performance in terms of RMSE for DT, RF, GBTs and LR for buildings 32, 617, 2713 and 459.

a building, as explained in Algorithm 1. To indicate the accuracy of the short-term load forecasting model, RMSE metric [32].

Only the forecasting of 1 hour is depicted in Figure 5.4. Interesting is that not all models have the same predictive performance on a specific time series dataset. The reason for that lies in the characteristics of time series dataset. While DT achieves the best performance for building 32, LR introduces the smallest RMSE for forecasting the load of building 617.

5.3.2. Similarity-based Clustering Analysis

The experiments are conducted on datasets of 60 energy time series selected from Energo+ presented in 2.4.1. These time series are measurements of the energy load from 60 institutional buildings at a granularity level of 1-hour for 5 years. The original granularity level is 15 minutes, therefore we aggregate the readings in 1-hour before starting this group of experiments. Before applying the DSTMF methodology to extract the meta features, a preprocessing procedure is applied to cleanup and harmonize the energy time series datasets. The procedure includes detecting gaps and NA values and performing outlier analysis.

According to the DSTMF methodology presented in Section 5.2.1, the input time series datasets which had a time resolution of 1-hour, are then aggregated to daily, monthly, weekend, workday, workday_sum, winter, summer, spring and autumn. For each aggregated dataset, the five-number summary and the arithmetic mean are calculated resulting in 6 meta features. As a result, 60 meta features are extracted for each input time series dataset formulating a final meta feature matrix of 60 columns and 60 rows.

As mentioned before, it is a great of importance to perform a benchmark evaluation, in which the DSTMF's meta features are compared to state-of-the-art meta features of other approaches existing in literature and applied in meta learning model selection for time series forecasting. For this benchmark evaluation, the four different groups presented in 5.2.2 are used. To this end, 2 S.F., 8 St.F., 2 I.F., 26 T.F. and 60 DSTMF's meta features are extracted for the 60 time series energy load datasets forming the final meta feature matrix.

After extracting the meta features for our input load time series datasets, a similarity-based clustering analysis is performed to measure the accuracy of the extracted meta features in representing the behavior of those datasets. The goal of clustering is to identify similarities between unlabeled datasets by organizing the datasets into homogeneous groups with the help of some similarity-criteria where this criteria value is minimized for datasets within the same group and the between-groups dissimilarity is maximized.

The main idea behind clustering lies in the fact that, if a set of meta features accurately captures the characteristics of a set of datasets, then a clustering algorithm applied on those datasets and their corresponding meta features separately should arrive at the same cluster classification. Or more precisely, the meta features that are grouped together in one cluster should represent the datasets which are also located together in one cluster for applying the same clustering algorithm.

To ensure that the clustering analysis works as expected, an appropriate clustering algorithm is needed. Our decision for choosing the right clustering algorithm is based on internal measures which use the intrinsic information in the time series datasets to assess the quality of the clustering. Internal measures entail the connectivity, the silhouette coefficient and the Dunn index. Briefly, connectivity indicates the degree of connectedness of the clusters, as determined by the k-nearest neighbors algorithm. The connectedness corresponds to what extent items are placed in the same cluster as their nearest neighbors

Table 5.1.: Internal measures of applying 10 clustering algorithms on time series datasets.

Clustering Category	Hierarchical Clustering	k-means	PAM	DIANA	CLARA	SOTA	Agnes	Model-based	SOM	FANNY
Connectivety	43.1948	19.2528	34.8175	45.9135	34.1147	51.7861	25.8881	60.8845	64.4861	54.4458
Silhouette	0.1428	0.2819	0.1511	0.1313	0.1527	0.1184	0.2015	0.0984	0.0901	0.1004
Dunn Index	0.4955	0.7319	0.5143	0.4978	0.6126	0.3998	0.7009	0.2011	0.1184	0.3126

in the data space. The connectivity ranges from 0 to infinity and should be minimized for better clustering. Silhouette width and Dunn index combine measures of compactness and separation of the clusters. The values of silhouette width range from -1 as poorly-clustered observations to 1 as well-clustered observations. The Dunn index is the ratio between the smallest distance between observations not in the same cluster to the largest intra cluster distance. It has a value between 0 and infinity and should be maximized for better clustering.

Normalization is an essential step before clustering. This is due to the fact that normalization controls the variability of the dataset by converting it into a specific range to generate good quality clusters and improve the accuracy of clustering algorithms.

Table 5.1 summarizes our results after applying a wide range of clustering algorithms [177], namely hierarchical clustering, k-means, PAM, DIANA, CLARA, SOTA, Agnes, Model-based, SOM and FANNY, on our 60 normalized energy load time series datasets. From the aforementioned results, the clustering algorithm that outperformed all others is chosen to evaluate the accuracy of meta features. Overall, according to Table 5.1, the k-means clustering algorithm performed best with regard to the connectivity, the silhouette coefficient and the Dunn index, therefore, it was chosen as the clustering algorithm for our experiments to evaluate the accuracy of meta features.

However, determining the optimal number of clusters in the k-means algorithm is a great challenge. For the presented work, the statistical testing method, called gap statistic proposed by Tibshirani et. al. in [159] is used to determine the optimal number of clusters. The gap statistic compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data points. The estimation of the optimal number of clusters will then be the value that maximizes the gap statistic. As a result, k = 8 is considered as an optimal number of clusters for the k-means clustering algorithm. Based on the aforementioned configurations, the clustering analysis is performed on the 60 normalized energy load time series datasets and their corresponding meta features with the following cluster groups: simple, statistical, information-theoretic, time series, DSTMF, all without DSTMF and all meta features.

Figure 5.5 illustrates the distribution of the 60 time series datasets on 8 clusters after applying the k-means clustering algorithm. As each dataset is represented by multiple meta features, k-means clustering algorithm will perform Principal Component Analysis (PCA) and plot the data points in 2 dimensions corresponding to the first two principal components that explain the majority of the variance in data.

The Clustering Error (CE) is defined based on Equation (5.1).

$$CE = \sum_{i=0}^{n} \frac{CW_i}{CT_i} \tag{5.1}$$

Where:

- CW_i: the number of the dataset that are wrongly clustered in cluster i.
- CT_i: the total number of datasets in cluster i.
- n: the total number of clusters.

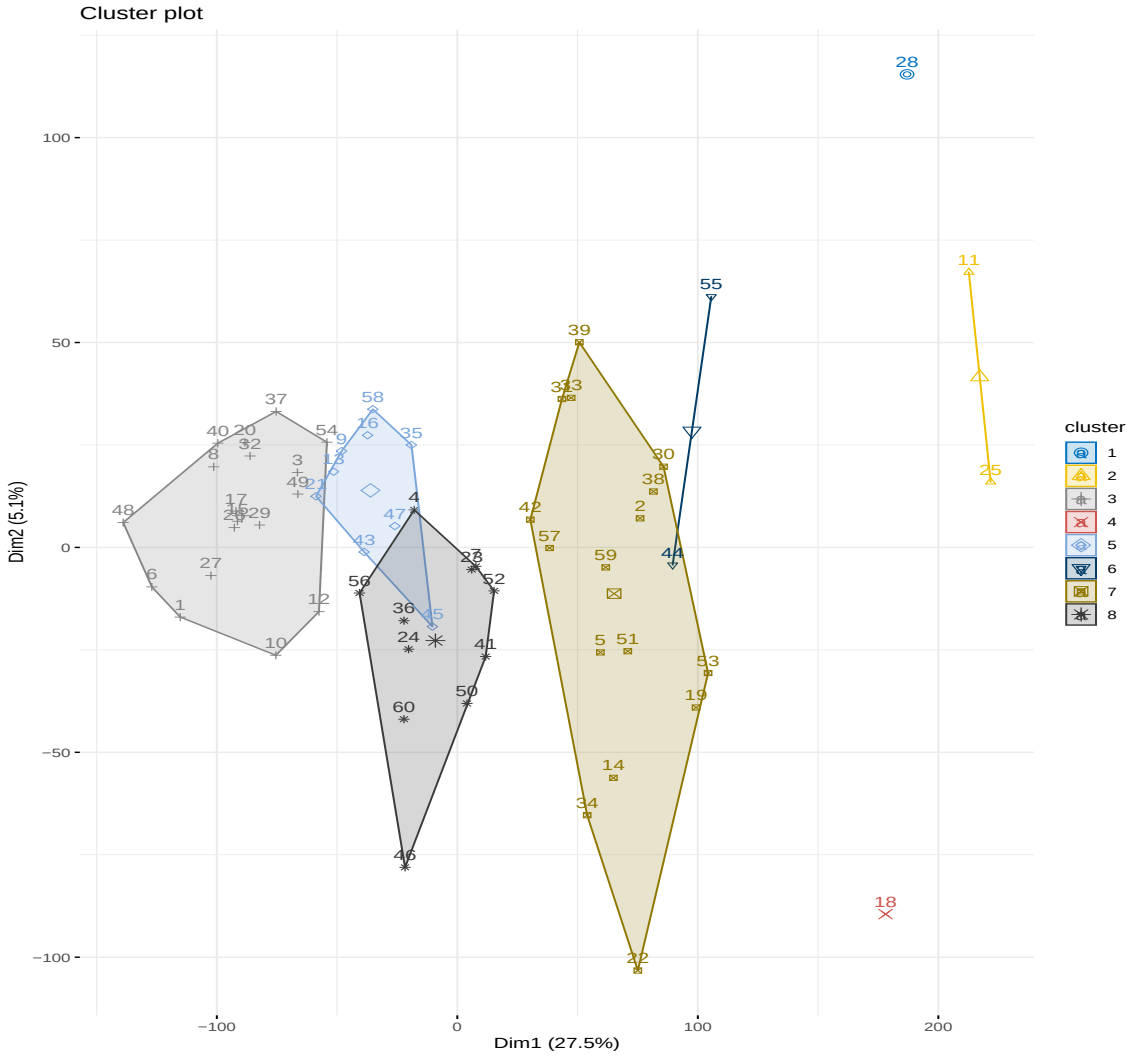


Figure 5.5.: Result of applying the k-means algorithm with 8 clusters on the original input time series datasets.

To understand how the error is calculated, consider the cluster number 5 (depicted in light blue in Figure 5.5) as an example. In this cluster, the datasets number 58, 16, 35, 9, 13, 21, 47, 43 and 45 are grouped together in one cluster. Assume that the k-means

clustering algorithm is applied on the group of statistical meta features. As a result, only the meta features datasets number 58, 16, 35, 9 and 43 are grouped together in one cluster and the others are clustered in different clusters. The majority of meta features datasets that are grouped together are taken as a reference of correct clustering and all others are considered as wrongly clustered. According to this, 4 out of 9 meta features datasets are in the wrong clusters. This calculation is repeated for the other clusters to discover the meta features datasets which are clustered in wrong clusters. This error is accumulated to find the final error of clustering.

Table 5.2.: Clustering error for different groups of meta features.

Clustering Category	Clustering Error
Simple meta features	80%
Statistical meta features	70%
Information-theoretic meta features	56%
Time series meta features	41%
DSTMF's meta features	39%
Simple + Statistical + Information-theoretic + Time series meta features	31%
All meta features	18%

Table 5.2 shows the comparison of results for applying the clustering on different groups of meta features. Note that both simple and statistical meta features characterize time series datasets only to a low extent. This can be seen in clustering error of 80% and 70% for simple and statistical meta features respectively. An improvement of 14% is achieved by information-theoretic meta features. An acceptable capturing of the characteristics of time series datasets is obtained by time series meta features, where the clustering error is 41%. DSTMF's meta features outperform all previous meta features by achieving a clustering error of 39%.

A better clustering with an error of 31% is attained when using simple, statistical, information-theoretic and time series meta features together. The major effect of our new meta features can be clearly seen when applying clustering on all meta features including DSTMF' s meta features, whereby a further improvement of 13% is realized with a clustering error of 18%. This gives good evidence for the effectiveness of our proposed meta features in capturing the deeper properties of time series datasets.

5.3.3. Predictive Performance of Meta Learner: Original Representation of Meta Features

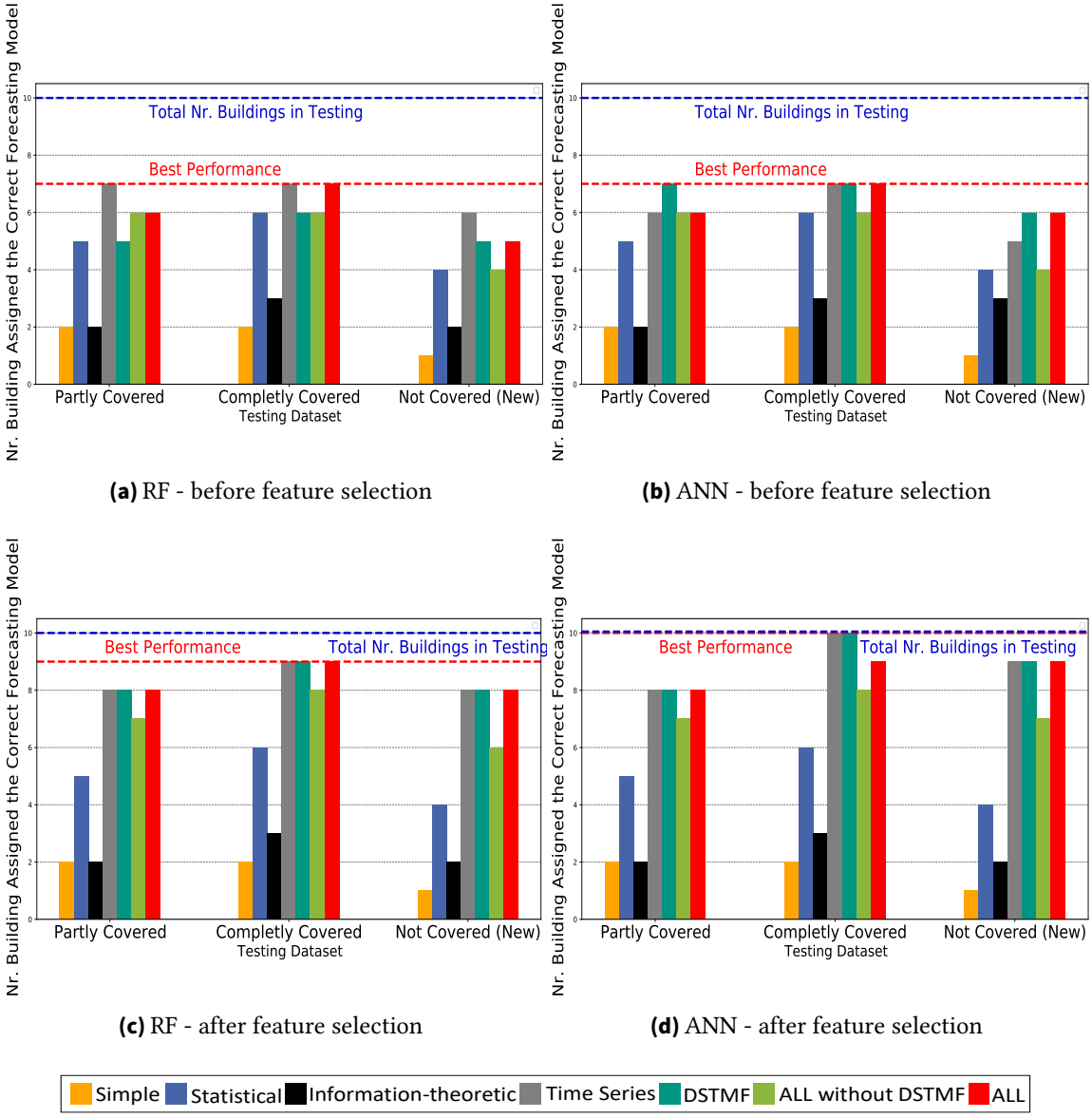
In this part of the evaluation, the experiments are conducted on the dataset presented in Section 5.2.2. We extracted 128 meta features categorized in seven different groups to describe the energy time series datasets. To discover the association between the

meta features and forecasting models explained in Section 5.3.1, Random Forest (RF) and Artificial Neural Network (ANN) are employed as meta learners. For achieving the best predictive performance of the meta learner, it is important that the best set of hyperparameters for both meta learners is used. This is done by using the GridSearchCV technique [58], in which an exhaustive set of hyperparameter combinations is created and the model is then trained on each combination.

In this context, the used meta learner tries to achieve the goal of meta learning in selecting the most appropriate learning algorithm by formulating it as a supervised machine learning problem, in which a multi-class classification problem is solved. 140 time series datasets corresponding to 140 buildings are randomly selected from the original 200 buildings for training in this group of experiments.

For fair evaluation, three testing scenarios are taken into consideration. In the first one, the testing datasets represents 10 buildings that are partly covered by the training dataset. Secondly, the testing dataset represents 10 buildings that are completely contained in the training dataset. The last testing scenario defines the testing set by load datasets of 10 new buildings that are not covered by the training dataset. Concerning the difficulty for meta learner to recommend the best model, the second testing scenario represents a simple easy case compared to the last scenario that presents the difficult one. The experiments are repeated five times. Each time a new group of buildings is randomly chosen for training. After performing the tests, the mean of the results is computed and presented in Figure 5.6.

An overall comparison between the different categories of meta features is presented in this figure. This comparison is based on the number of buildings that have been assigned the correct short term load forecasting model. Generally, a relatively bad classification performance is introduced by both meta learners especially in the third testing scenario, in which a new group of buildings that are not covered by the training dataset is used as seen Figures 5.6a and 5.6b. This bad performance has two main reasons, namely the curse of dimensionality and multicollinearity between meta features. The negative effect of multicollinearity can be interpreted as some meta features are highly correlated in a way that can negatively affect the performance of the meta learner. This is due to the fact that multicollinearity problem can lead to a false interpretation of feature importance as the importance level of certain features can be underestimated or overestimated.



(a) RF - before feature selection

(b) ANN - before feature selection

(c) RF - after feature selection

(d) ANN - after feature selection

Figure 5.6.: Predictive performance of meta learning classification models in case of using Random Forest (RF) and Artificial Neural Network (ANN) as meta learners.

To overcome these problems, the proposed EMLS applies an efficient feature engineering procedure for identifying the main independent from each other meta features and delete all predictors that have no effect on the response variable. Pearson's correlation [112] is applied to find the highly correlated meta features in order to remove them. As seen in Figure 5.7, a relatively high correlation is found among meta features which negatively affected the predictive performance of the meta learner. Due to the large number of DSTMF meta features, we will show the correlation between 30 meta features.

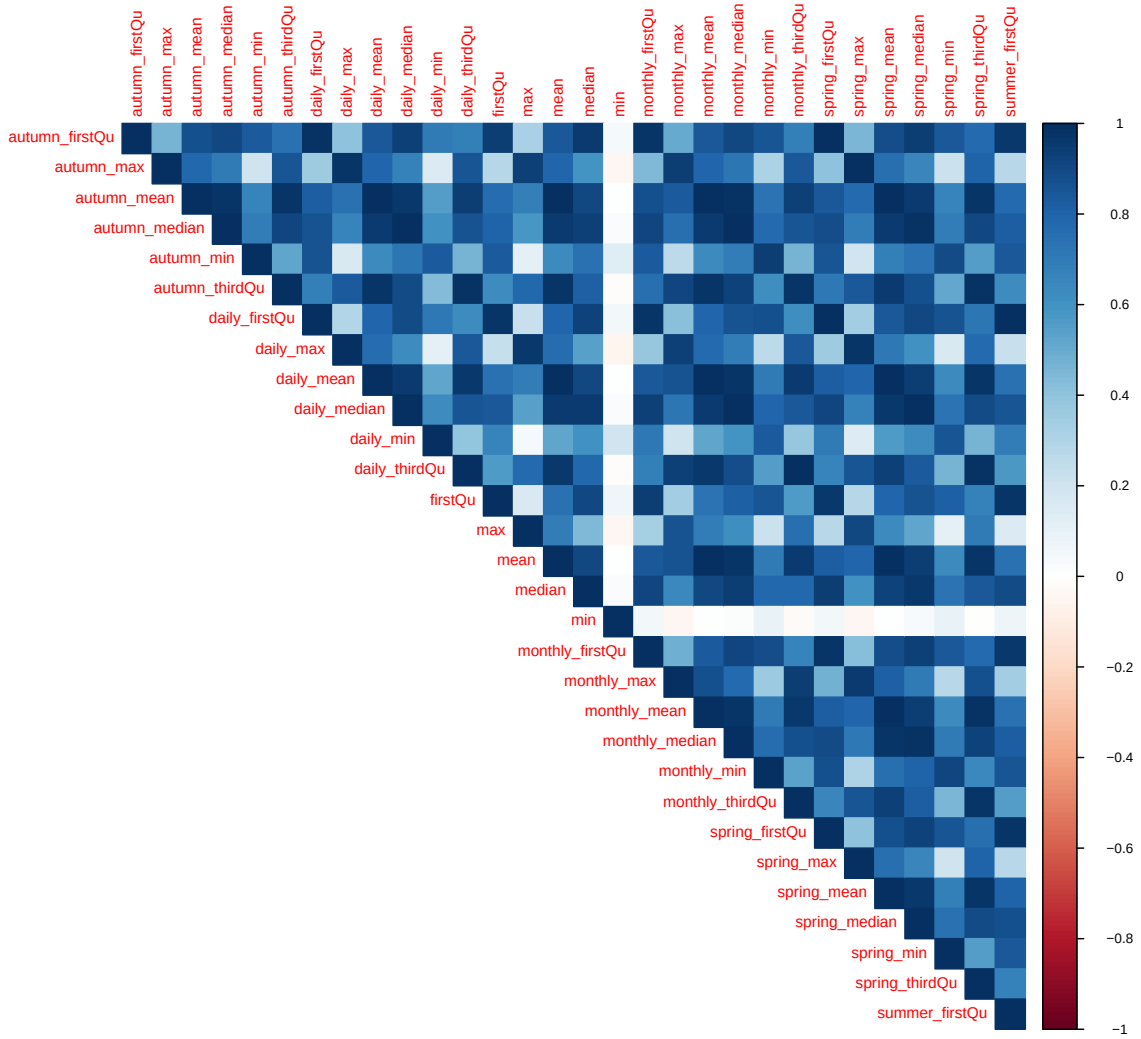


Figure 5.7.: Correlation matrix of 30 DSTMF meta features.

After discovering the highly correlated meta features, the next step is to remove them. To this end, we remove the meta features that are > 0.75% correlated. Figure 5.8 and 5.9 illustrates a part of the time series and DSTMF meta features after removing the highly correlated meta features.

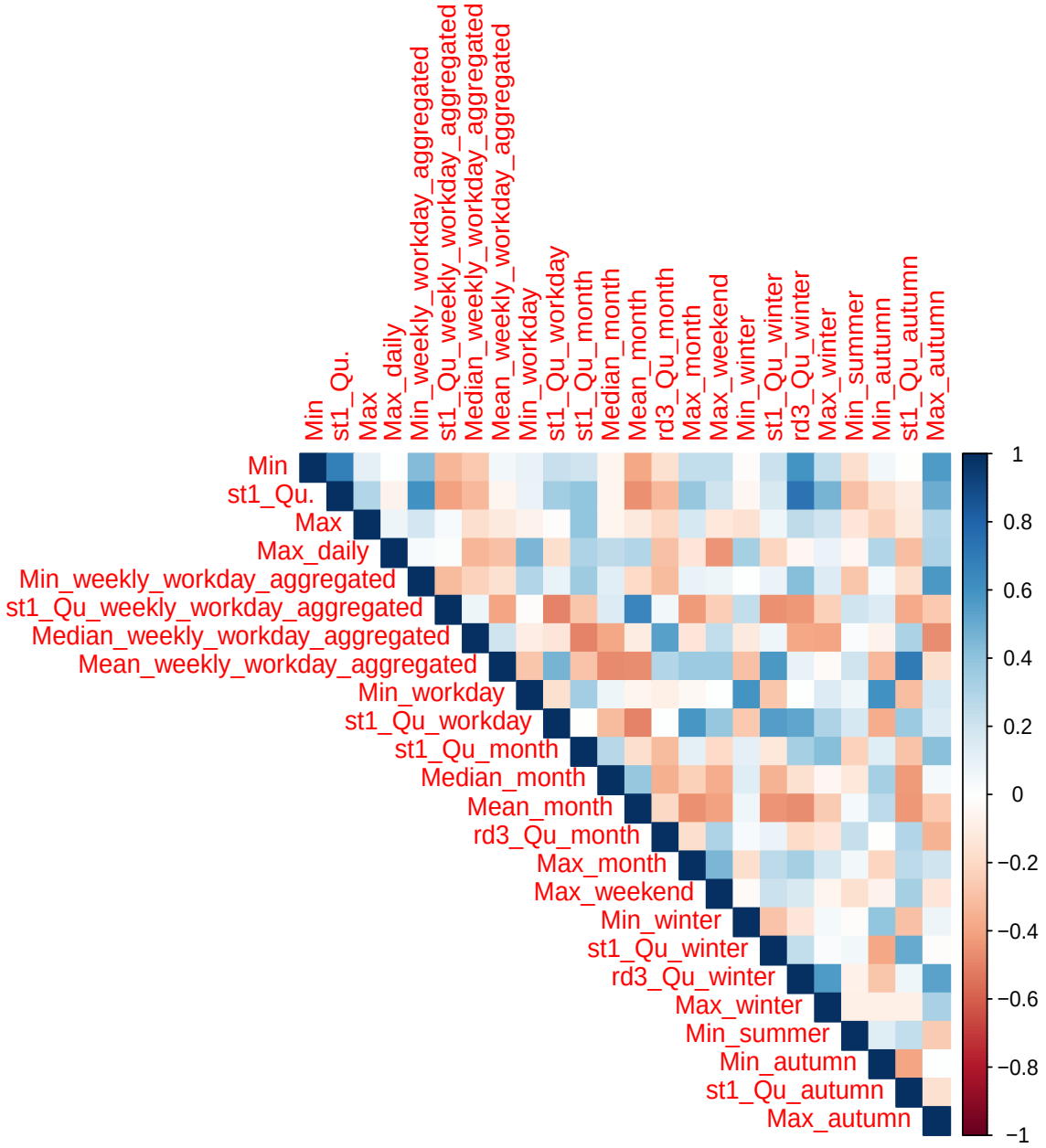


Figure 5.8.: A part of DSTMF meta features after removing the features that are > 0.75% correlated.

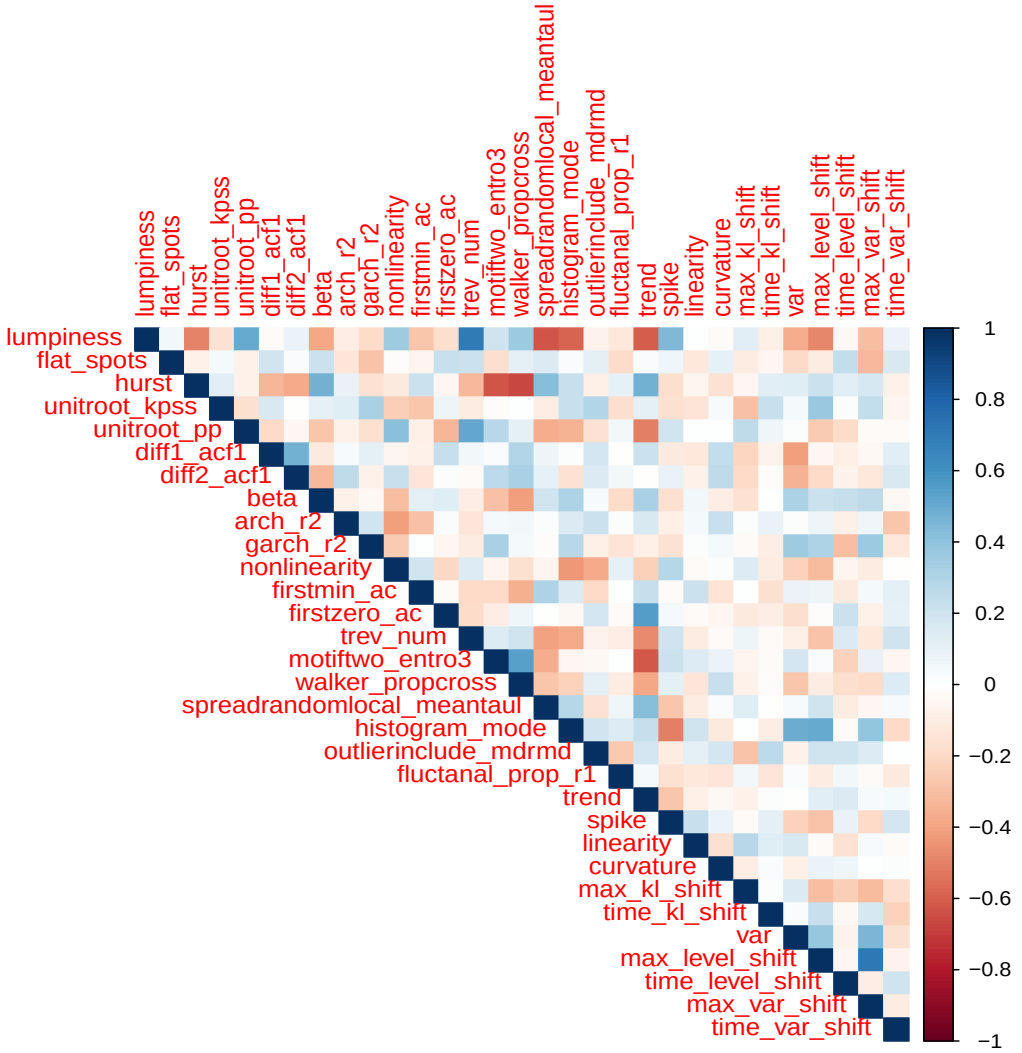


Figure 5.9.: A part of time series meta features after removing the features that are > 0.75% correlated.

As a result, the total number of meta features is reduced containing only the meta features that are not strongly correlated as seen in Table 5.3.

Table 5.3.: Different groups of meta features after removing the highly correlated meta features.

Meta Features Groups	Total Number of Meta Features	Number of Not Highly correlated Meta Features
Simple meta features	2	2
Statistical meta features	7	7
Information-theoretic meta features	2	2
Time series meta features	45	26
DSTMF meta features	72	44
Simple + Statistical + Information-theoretic + Time series meta features	56	35
All meta features	128	76

After removing the highly correlated meta features, we need to select the most important predictive features. To achieve that, Recursive Feature Elimination (RFE) is used. RFE is an automatic feature selection method that can be used to build many models with different subsets of a dataset and identify those attributes that are not required to build an accurate model. This technique begins by building a model on the entire set of features and computing an importance score for each one. The least important feature(s) are then removed, the model is re-built, and importance scores are computed again. Table 5.4 illustrates the features selected by RFE for each meta-feature group.

Before feature engineering and in the case of using RF as meta learner, the best predictive performance is achieved by time series meta features for the three testing scenarios as depicted in Figure 5.6a. In the second testing scenario, the best predictive performance is achieved by time series as well as all, the last category, as also depicted in Figure 5.6a. ANN showed the best predictive performance with the categories DSTMF, time series and all, as seen in the second testing scenario in Figure 5.6b. In this figure, DSTMF outperforms time series meta features in the first and third testing scenarios. As depicted in Figure 5.6c, EMLS is able to assign the correct forecasting model for 9 buildings in the second testing scenario when using time series, DSTMF and all meta features.

Interesting is that the whole number of buildings are assigned the correct model by using ANN with time series and DSTMF in the second testing scenario as seen in Figure 5.6d. Focusing on the third testing scenario, the most difficult one, the best predictive performance of assigning 9 correct models to 10 buildings is achieved in the case of using ANN for the categories time series, DSTMF and All as seen in Figure 5.6d.

Moreover and by comparing the last 2 categories of meta features, an improvement is generally noticed for all testing scenarios in case of using DSTMF as a part of meta feature space, namely category all, as depicted in Figures 5.6a, 5.6b, 5.6c and 5.6d. EMLS achieves an improvement of 10% to 30% after applying the feature engineering procedure as illustrated in Figure 5.6. Moreover, it is notable that a bad predictive performance is introduced by both RF and ANN for simple and information-theoretic meta features in all testing scenarios. This provides an evidence that these categories badly characterize time series datasets. Moreover, the proposed DSTMF represents a new comparative art of meta features against time series meta feature category as seen in Figures 5.6b, 5.6c and 5.6d. Furthermore, it is found that using ANN as a meta learner introduces better predictive performance in assigning the best forecasting model for energy time series datasets.

5.3.4. Predictive Performance of Meta Learner: Encoded Representation of Meta Features Using Autoencoders

In this part of our evaluation, we investigate the effect of encoding meta features on the predictive performance of meta learners in predicting the most appropriate forecasting model. The experiments are also conducted on the dataset presented in Section 5.2.2. We extracted 128 meta features categorized in seven different groups, namely simple, statistical, information-theoretic, time series, DSTMF, all meta features without DSTMF

Table 5.4.: The meta features selected by Recursive Feature Elimination (RFE) procedure for each group of meta features.

Meta Feature Category	Total Nr. of Meta Features	Selected Meta Features	Nr. of Selected Meta Features
Simple Meta Features	2	Nr. of samples and Nr. of attributes	2
Statistical Meta Features	7	Var, skewness, std, kurtosis, std1st_der, outlierinclude_mdrmd, histogram_mode	7
Information-theoretic Meta Features	2	Entropy, motiftwo_entro3	2
Time Series Meta Features	26	stability, x_acf10, unitroot_pp, x_acf1, ac_9, e_acf1, alpha, diff2x_pacf5, arch_lm, diff1_acf1, spreadrandomlocal_meantaul, diff2_acf10, diff2_acf1, time_level_shift, linearity, walker_propcross, flat_spots, unitroot_kpss, firstzero_ac, trend	20
DSTMF Meta Features	44	autumn_max, weekly_workdays_max, weekly_max, daily_max, winter_max, workdays_max, weekends_max, winter_thirdQu, winter_mean, daily_min, weekends_mean, weekly_weekends_median, weekends_median, daily_median, monthly_thirdQu, winter_firstQu, summer_max, autumn_thirdQu, monthly_mean, daily_thirdQu, workdays_median, workdays_mean	22
All Without DSTMF	35	stability, x_acf10, unitroot_pp, x_acf1, ac_9, e_acf1, alpha, diff2x_pacf5, arch_lm, diff1_acf1, spreadrandomlocal_meantaul, diff2_acf10, diff2_acf1, time_level_shift, linearity, walker_propcross, flat_spots, unitroot_kpss, firstzero_ac, trend, entropy, motiftwo_entro3, var, skewness, std, kurtosis, std1st_der, outlierinclude_mdrmd, histogram_mode, count	30
All Meta Features	76	stability, x_acf10, unitroot_pp, x_acf1, ac_9, e_acf1, alpha, diff2x_pacf5, arch_lm, diff1_acf1, spreadrandomlocal_meantaul, diff2_acf10, diff2_acf1, time_level_shift, linearity, walker_propcross, flat_spots, unitroot_kpss, firstzero_ac, trend, entropy, motiftwo_entro3, var, skewness, std, kurtosis, std1st_der, outlierinclude_mdrmd, histogram_mode, count ,autumn_max, weekly_workdays_max, weekly_max, , daily_max, winter_max, workdays_max, weekends_max, winter_thirdQu, winter_mean, daily_min, weekends_mean, weekly_weekends_median, weekends_median, daily_median, monthly_thirdQu, winter_firstQu, summer_max, autumn_thirdQu, monthly_mean, daily_thirdQu, workdays_median, workdays_mean	52

and all meta features including DSTMF. Only the most four comparative groups of meta features are included in this group of experiments, namely time series, DSTMF, all meta features without DSTMF and all meta features including DSTMF. This is due to their good performance as seen in Section 5.3.3.

An autoencoder neural network is an unsupervised deep learning technique [20] that employs backpropagation for setting the target values to be equal to the inputs [101]. Its architecture can vary between a simple FeedForward network, convolutional neural network and LSTM network, to name a few. A simple architecture is utilized here as the main goal of the current evaluation is to reconstruct the meta features in another powerful representation form capturing the useful meta information already included in meta features as depicted in Figure 5.10.

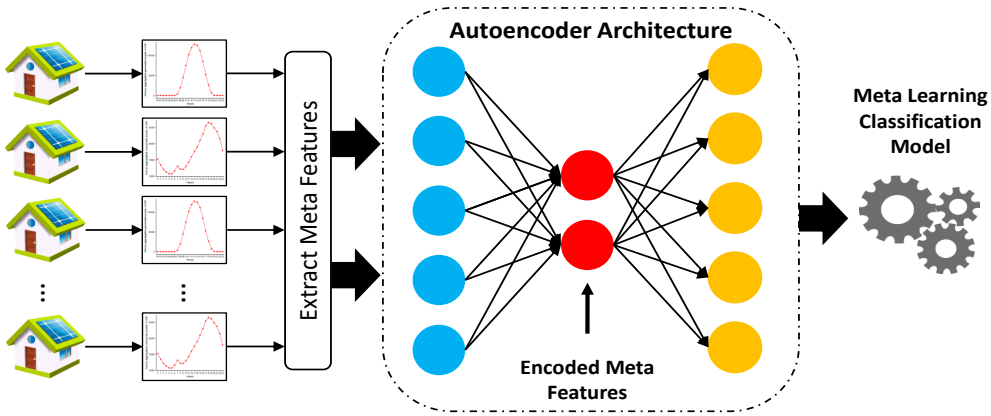


Figure 5.10.: Autoencoder architecture utilized for reconstructing the meta features in another representation form.

The parameter settings of autoencoders are as follows: the number of neurons in the last layer is equal to them in the first layer that corresponds to the number of input meta features we want to encode and the activation function is Rectified Linear Unit (ReLU). The reason for using ReLU lies in the fact that unlike classical activation functions such as "Tangens Hyperbolicus" [66] and "sigmoid logistic" functions [78], ReLU allows exact zero values easily which makes it a good candidate for encouraging sparsity of representation [51].

The autoencoder reconstructs its inputs by minimizing the difference between the input meta features χ and the output Y instead of predicting the target value for given inputs. To achieve that, two parts are required, namely encoder and decoder, which can be defined as transitions Φ and Ψ in Equation (5.2):

$$\Phi : \chi \rightarrow F, \psi : F \rightarrow \chi \tag{5.2}$$

The encoder stage of an autoencoder takes a row x in the input meta feature matrix χ and maps it to another representation form $h \in F$ in Equation (5.3):

$$h = \sigma(Wx + b) \tag{5.3}$$

where:

- h represents a latent variable, or latent representation of the input meta features.
- σ is the activation function.
- W is a weight matrix and b is the bias vector, whereby weights and biases are initialized randomly and then updated iteratively by feedforward backpropagation.

After that, the decoder stage maps h to another representation form $x^{'}$ of the same shape of input meta feature x as presented in Equation (5.4):

$$x^{'} = \sigma^{'}(W^{'}h + b^{'}) \tag{5.4}$$

The autoencoder is then trained to minimize the reconstruction error of meta features, referred as loss according to Equation (5.5):

$$L(x, x^{'}) = \left\| x - x^{'} \right\|^2 = \left\| x - \sigma^{'}(W^{'}(\sigma(Wx + b)) + b^{'}) \right\|^2 \tag{5.5}$$

After integrating the autoencoder into the system, the meta learning use case presented in EMLS is modified according to the methodology of EEMLS and evaluated to discover the effect of meta features encoding on the predictive performance of the meta learner.

As a meta learner, ANN is used. To get consistent results, the experiments are repeated five times. Each time a random set of building load datasets are selected for training ANN meta learner and the mean values of the evaluation results are depicted in Table 5.5. Despite no information compression is achieved by autoencoder in Setup Configurations (SCs) 1, 5, 9, 13 and 17, as the number of neurons in the middle layer is equal to them in the first input layer, an acceptable performance is obtained. An improvement is achieved in SCs 2, 6, 10, 14 and 18, whereby a compression ratio of 50% is performed by the autoencoders. For further compression, no improvement is noticed as a part of meta information is lost.

Generally, the best performance is introduced by the T.F. and DSTMF meta features. As seen in Table 1, this performance is changing according to the used setup configurations, where we reduce the total number of buildings used for training to accurately measure the strength of meta features in capturing the deep characteristics of time series datasets. While in some cases, time series meta features outperforms DSTMF, DSTMF introduces better performance especially for a small size of training set. Precisely, it can be clearly seen that for a number of buildings >= 64, T.F. meta features outperforms DSTMF as seen in SC from 1 to 8. As the number of buildings in the training set is decreased, the task is more difficult for meta learners to learn and assign the correct models. In these situations, DSTMF outperforms all other categories.

Table 5.5.: Setup configurations and evaluation results of EEMLS.

Setup Configuration (SC)				Meta Features Category			
Nr.	**Training Size**	**Testing Size**	**Nr. Neuron in Middle Layer**	**T.F.**	**DSTMF**	**All without DSTMF**	**All**
1	118	20	All meta features	19	19	19	19
2	**118**	**20**	**50 % of meta features**	**20**	**20**	**20**	**20**
3	118	20	25% of meta features	20	18	17	20
4	118	20	2 meta features	19	18	17	18
5	64	20	All meta features	19	18	18	19
6	**64**	**20**	**50 % of meta features**	**20**	**20**	**19**	**20**
7	64	20	25% of meta features	19	18	18	19
8	64	20	2 meta features	18	17	13	15
9	32	20	All meta features	17	17	16	17
10	**32**	**20**	**50 % of meta features**	**18**	**19**	**17**	**18**
11	32	20	25% of meta features	16	17	15	17
12	32	20	2 meta features	16	17	13	15
13	16	20	All meta features	17	18	14	16
14	**16**	**20**	**50 % of meta features**	**18**	**19**	**15**	**18**
15	16	20	25% of meta features	15	16	14	16
16	16	20	2 meta features	15	16	14	15
17	8	20	All meta features	17	18	13	15
18	**8**	**20**	**50 % of meta features**	**17**	**18**	**14**	**15**
19	8	20	25% of meta features	16	17	13	14
20	8	20	2 meta features	15	16	10	11

Overall, an improvement is achieved between the last two categories, in which the DSTMF meta features support meta learners with more useful meta information about time series datasets. This has two main reasons. On the one hand, the large amount of useful meta information contained in DSTMF meta features. On the other hand, the power of the autoencoder to encode the useful meta knowledge contained in the extracted meta features in a new efficient representation form.

By using autoencoders, the system is able to handle the multiclass classification problem of meta learning by a few number of examples in the training set, namely 32, 16 and even 8 examples introducing a good acceptable predictive performance in assigning the correct short term load forecasting model for a testing set of 20 new buildings. To this end and for a very few number of training examples, the proposed DSTMF outperforms T.F. meta features leading the meta learning classification model to achieve a better predictive performance, as seen in Table 5.5.

Consequently, the present thesis introduces a new meta learning approach, in which only a few number of time series datasets is required to build an efficient meta learning system for time series model selection without the need for simulating new time series in the case of a few available examples in the training set. Another advantage of encoding is that the proposed EEMLS introduces a good predictive performance not only with a small size of training set but also with a few number of features required to train the meta learning model, namely 25% of meta features and two meta features as seen in Table 5.5. This shows that EEMLS definitely produces very simple but effective models for performing energy time series model selection.

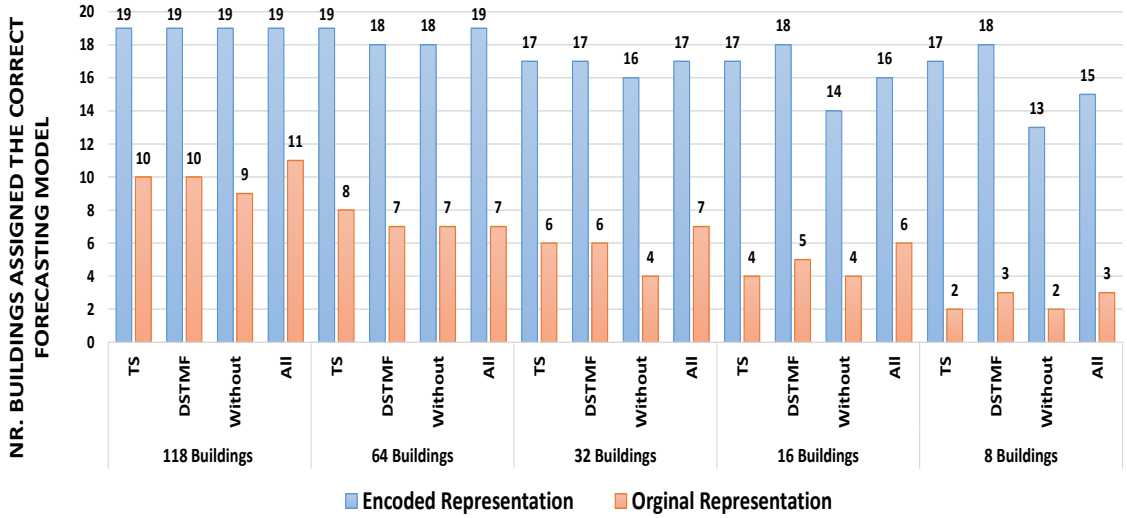


Figure 5.11.: The predictive performance of ANN meta learner in case of encoded and original representation of meta features.

To verify the positive effect of meta feature encoding, it is of great importance to compare the predictive performance of the ANN meta learner with its performance in the case of using the original representation form. Concerning the encoded meta features, only the setup configurations containing the whole number of meta features are taken into account, namely 1, 5, 9, 13 and 17. By the encoded representation of meta features, the ANN meta learner is able to assign the correct forecasting model to the majority of buildings in the testing set as seen in Figure 5.11. This number decreases as the size of the training set is decreased. Moreover, the encoded meta features outperforms the original meta features as a larger number of buildings have been assigned the correct models.

Experiments show the tendency that encoding meta features is a good solution to alleviate the concern that solving a multiclass classification problem with a very few number of examples will lead to very bad performance. Consequently, in the case of only a few number of time series datasets available for training the meta learner, it is highly recommended to encode meta features by autoencoder for better utilization of meta information contained in them as seen in Figure 5.11.

5.4. Summary

In this chapter, Descriptive Statistics Time-based Meta Features (DSTMF) as a new methodology for characterizing the behavior of time series datasets with meta features to achieve a more accurate model selection for time series energy load forecasting was presented and evaluated. We started in Section 5.1 by describing the problem statement we aim to solve. In Section 5.2, the general solution methodology was presented. Our methodology included proposing Descriptive Statistics Time-based Meta Features (DSTMF) to describe energy time series datasets, and presenting Energy Meta Learning System (EMLS) as well as Encoded Energy Meta Learning System (EEMLS) for instrumenting the extracted meta features in building an efficient meta learning classification model. The main difference between EMLS and EEMLS lies in the utilization of unsupervised deep learning to encode the extracted meta features using autoencoder.

We evaluated the accuracy of DSTMF in characterizing energy time series datasets in Section 5.3 in three different groups of experiments. In the first group of experiments, we applied similarity-based clustering analysis to measure the efficiency of DSTMF against state-of-the-art meta features existing in the literature. We found that DSTMF outperformed time series, information-theoretic, simple and statistical meta features in characterizing energy time series datasets. In the second group of experiments in Section 5.3.3, we compared the efficiency of DSTMF in terms of the predictive accuracy of meta learners in assigning the best forecasting model to the input energy time series datasets. To achieve that, random forest and artificial neural networks were used as meta learners. Different groups of meta features were extracted. The accuracy of meta learners was then measured in the case of using DSTMF as meta features and compared to the cases of using the other groups of meta features. This group of experiments was carried out in two points of view, namely with and without applying feature engineering procedure. Generally, DSTMF introduced better performance in characterizing energy time series datasets than the other groups of meta features. An improvement of 10% to 30% was acquired after applying the feature selection procedure.

In the last group of experiments, we discovered the accuracy of DSTMF in characterizing energy time series meta features in the case of using an autoencoder. We used an autoencoder to encode meta features in another representation form. Generally, we found that the predictive performance of meta learner in assigning the best forecasting model for buildings outperformed the case of using original representation of meta features.

6. Generating Efficient Meta Examples for Energy Time Series Model Selection

Meta learning approaches have been proposed to address the issue of model selection [81][151][70][122] by formulating it as a classification problem whereby each dataset is assigned an adequate model. Considering meta learning as a multi-class data-driven approach and in order to perform well, meta learning systems require a sufficiently large training set to learn from. When the amount of originally acquired data is not enough to train a well-performing meta learner, the meta learning model will overfit on the training data and generalize badly [83][117].

By increasing the training set size, meta learning performance can be improved. The reason for that lies in the fact that the predictive performance of a classification model highly depends on the examples available for training not only in number but also in their diversity. To this end, there has been a lot of recent work published in literature presenting methodologies and approaches for generating new time series datasets which could be used as training dataset [47][156][121].

In this chapter, we present a new approach for enhancing the predictive performance of meta learners by adding newly generated meta examples to the training set with some new but simple generation strategies. We start in Section 6.1 by presenting the problem statement including the research question we aim to answer. After that, our concept to enhance the predictive performance of the meta learning classification model in recommending the best forecasting model is introduced and discussed in detail in Section 6.2. Thereafter, we evaluate the effect of the inclusion of the newly generated meta examples on the predictive performance of the meta learning classification model in Section 6.3. In our evaluation study, two representations forms of meta features are taken into account, namely, original and encoded representation of meta features. The research contributions presented in this chapter were the main topics of our paper in [146].

Parts of this chapter are reproduced from:

- S. Shahoud, M. Winter, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2021). "An extended Meta Learning Approach for Automating Model Selection in Big Data Environments using Microservice and Container Virtualization Technologies". In: Internet of Things, vol. 16, p. 100432. doi: 10.1016/j.iot.2021.100432.

6.1. Problem Statement

Meta learning for energy time series forecasting model selection is based on accumulating the meta knowledge over several energy time series datasets and then using such acquired knowledge to recommend the best model for a new energy time series dataset. Each dataset is represented by a meta example consisting of a set of meta feature values and a label that corresponds to the most suitable forecasting model for this time series dataset. As mentioned before, the process of learning in this context is essentially considered as a multi-class classification problem in which the meta learner tries to learn the mapping between the meta knowledge and the corresponding forecasting models.

Like many other machine learning problems, a multi-class classification problem also has the challenge that it needs an adequate domain data set in size and diversity for training. The better (diverse) and more examples are available for the learning process, the better the classification performance will be. To face this challenge, many approaches are proposed in literature to generate synthetic time series dataset or simulate new ones[65][188][64].

In this chapter, We first develop a meta learning system which predicts the best one-day-ahead forecasting algorithm for a given solar power generation time series. We further answer the research question RQ3 ‘ How to enhance the performance of automated model selection in the context of energy by creating appropriate learning datasets?” by generating additional meta examples which we can be added to the original training set in order to create extended training sets. This is achieved by creating new time series, which can be transformed into said meta examples. The additional meta examples will reduce variance and overfitting and ultimately result in better classification performance by the meta learner. Our concept incorporates many different methodologies to generate new time series datasets, namely time-based, weather-based and a combination between them to increase the number of meta examples which in turns enhances the predictive performance of the meta learning classification problem.

6.2. Proposed Solution

In order to enhance the predictive performance of our meta learning multi-class classification problem, if our training dataset is too small, more meta examples are required. To augment the training dataset, we propose a new simple approach for generating new time series datasets using the existing datasets without the need of instrumenting complex simulation models or synthetically generating new time series. Newly generated time series data sets are automatically added to the existing training set and transformed into meta examples by extracting the relevant meta features and determining the label of the best forecasting model. The new resulting meta examples will then be used to train the meta learner again on the bigger training data set and enhance its performance in recommending the best forecasting model for a new time series dataset.

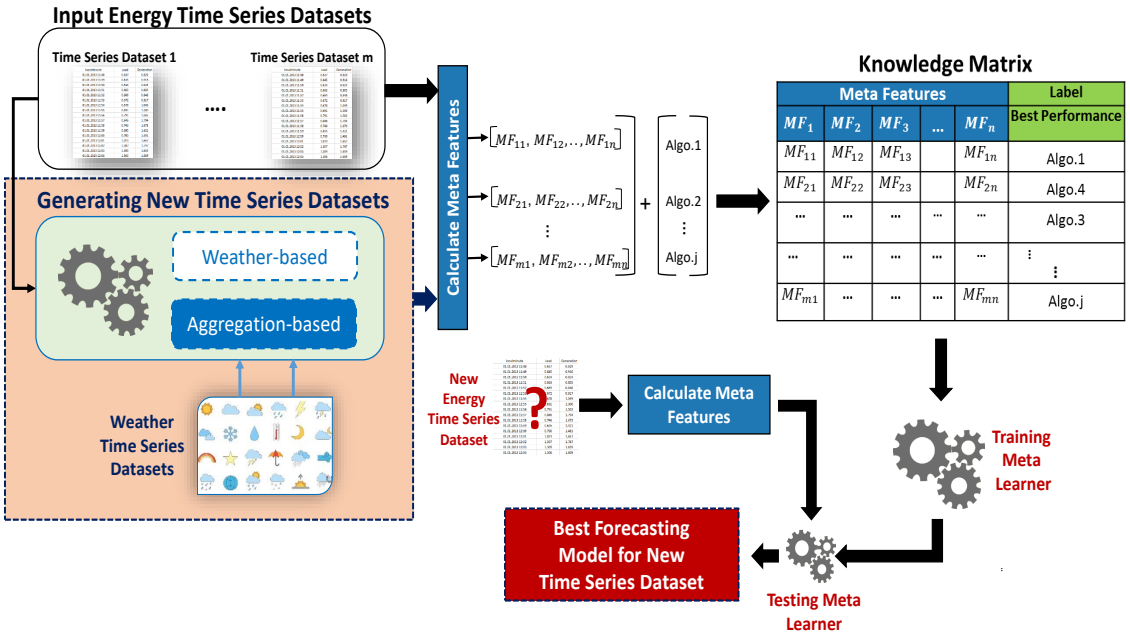


Figure 6.1.: Methodology of enhanced meta learning approach, adapted from Chapter 5 with an additional component for generating new time series datasets.

To incorporate the new generated features into our solution, we extend the main meta learning methodology presented in Section 5.1 as shown in Figure 6.1. Our approach for generating new time series datasets uses two methodologies, namely aggregation-based and weather-based. Both of them will later be compared to the model-based approaches, in which new time series are simulated. As shown in Figure 6.1, the aggregation-based approach uses the input energy time series datasets to generate new ones by aggregating values into different aggregation levels as detailed in 6.2.3. The weather based approach groups (parts of) datasets according to their behavior under certain weather conditions and constructs new ones by varying the weather situations as detailed in 6.2.2. All resulting energy time series datasets are passed to extract the relevant meta features such as information-theoretic, statistical, time series and DSTMF meta features. We use these meta features as features in the knowledge matrix. Then, we build a benchmark of power generation forecasting models and set the best one as a label for each time series dataset in the knowledge matrix.

To find the mapping between the extracted meta features and the best forecasting model, another ML algorithm is trained on the resulting knowledge matrix. As explained before, for a new energy time series dataset, the newly trained meta learner will then recommend the best forecasting model. The main goal of our concept is to increase the number of meta examples used for training the meta learner without the need for simulating new ones. To evaluate the effect of our new generated meta examples on the predictive performance of the meta learner, an in-depth analysis study is provided in Section 6.3. In this evaluation, we compare our proposed approaches with the model-based approach existing in literature for generating new time series datasets. Before presenting the three different approaches

for generating new time series dataset, an overview of the used solar power generation dataset is provided.

6.2.1. Dataset

Ausgrid solar home electricity data presented in Section 2.4.2 is used in this chapter to realize the new concepts for generating new meta examples. Even though the state-owned Australian energy provider Ausgrid applied the main data quality criteria on the gathered ausgrid solar home electricity data, some preprocessing steps are necessary before performing forecasting and afterwards meta learning. We start by performing a visual analysis on ausgrid solar home electricity data to recognize the time series datasets that contain big gaps in order to ignore them in our evaluation. While performing this procedure, We recognized that the gaps ranging from a couple of observations up to multiple months as seen in Figure 6.2. Such datasets with large gaps are discarded and not taken into consideration in our work as will be seen later in this section.

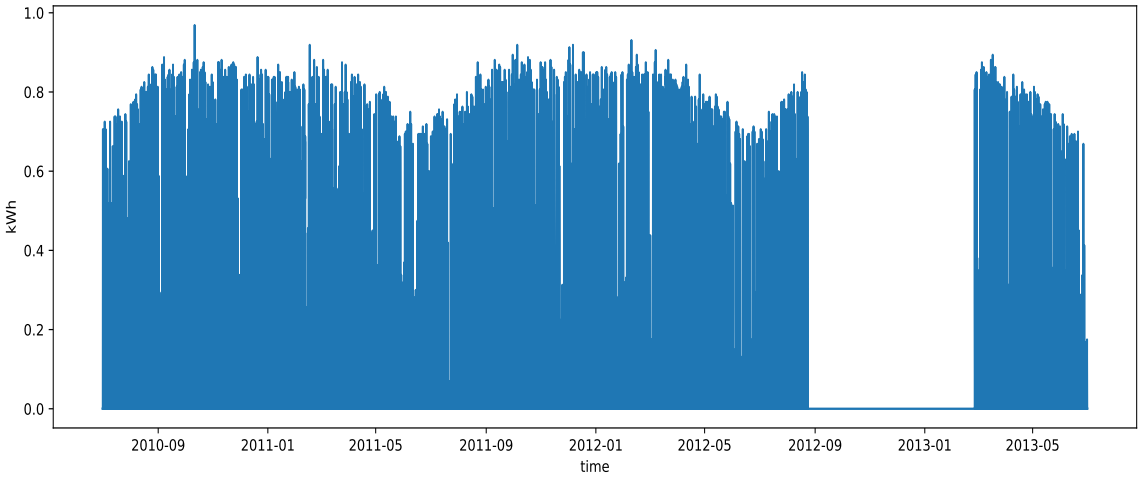


Figure 6.2.: Methodology of enhanced meta learning approach.

The next step is to detect the range of time series values that is taken into consideration in each day. To select the best range of time series values during the day to be considered in our forecasting, we divide every day into 3 sections as seen in Figure 6.3. Let S be the period ranging roughly from the daytime of the latest sunrise to the earliest sunset in a year in Australia. This interval ranges from 8 am to 4:30 pm and the solar systems are expected to always generate positive amounts of power during this time all year round, even if it is raining. Let D denote the period where it depends on the season, whether the solar systems generate power. Let N be the period where the sun (almost) never shines.

The different periods are illustrated for a winter and a summer day in Figure 6.3. The blue values correspond to a day that was recorded in the summer. The black curve corresponds to a day that was recorded in winter, where the sun goes down earlier. Period N is highlighted in red. Period D is orange and period S is highlighted in green. Note that

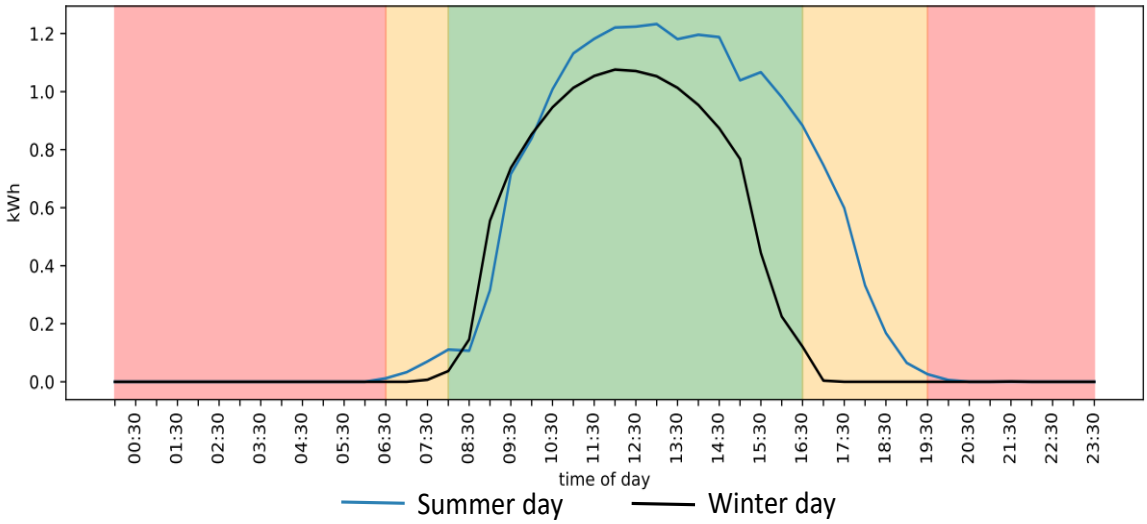


Figure 6.3.: Segmentation of daytime into different periods.

each data point represents an interval, e.g. the data point at 4:30pm represents the power generated from 4:30pm to 5pm.

Period S contains 17 observations and ranges from 8am to 4:30pm. Period D contains 9 observations and ranges from 6:30am to 8am and from 4:30pm to 7:30pm. Period N contains 20 observations ranging from 7:30pm to 6:30am. In total, these 3 periods consist of 48 observations. We remove all the data points in period N from our time series datasets. The time series values should always be zero or very close to zero during this time. Not removing those data points will slow down the forecast models computationally and also exhaust their learning capacity unnecessarily. So after removing period N, we have periods D and S, which encompass 26 observations in each day.

After segmentation, we now focus on period S to identify missing observations which present themselves as zero values. This is due to different reasons. We avoid using period D, because the fact that the zero values which can regularly occur based on seasonality would complicate the process of differentiating a missing from a correct observation. For period S, this is not the case and zero values are a clear sign of a missing observation. Furthermore, period S is the most important time frame that we are most interested in forecasting, because it is the period where most power is generated. Every time series contains 18632 total observations inside period S. The missing observations are ranged between 25 and 8943 points for every time series.

As a result from the visual analysis performed in the previous step, no negative values were found in any time series. In contrast to that, many high values as outliers were noticed. A Hampel filter is used to detect outliers in order to be removed from time series datasets. If the number of bad observations in a day is too high, interpolating the remaining values of the day will not yield good and realistic results. Thus, we label such days as unusable. We use the results from the previous steps and a day is marked as unusable, if it has more than three bad observations in succession. Having identified the unusable days, we now

determine whether or not to use a time series or to impute data on unusable days. If a time series has too many unusable days, we remove it from the dataset instead of replacing the missing data points, because we don't want to distort the time series too much. In this work, we decided to remove the time series from the dataset which had 40 or more unusable days. Consequently, 62 time series were removed and the remaining 238 time series are used in our work.

Considering the 238 time series dataset, we need no to fill the missing values. To achieve that, two approaches have been applied depending on the number and the range of missing values in each time series dataset. On one hand and for single missing values, we applied linear interpolation [104] as seen in Figure 6.4.

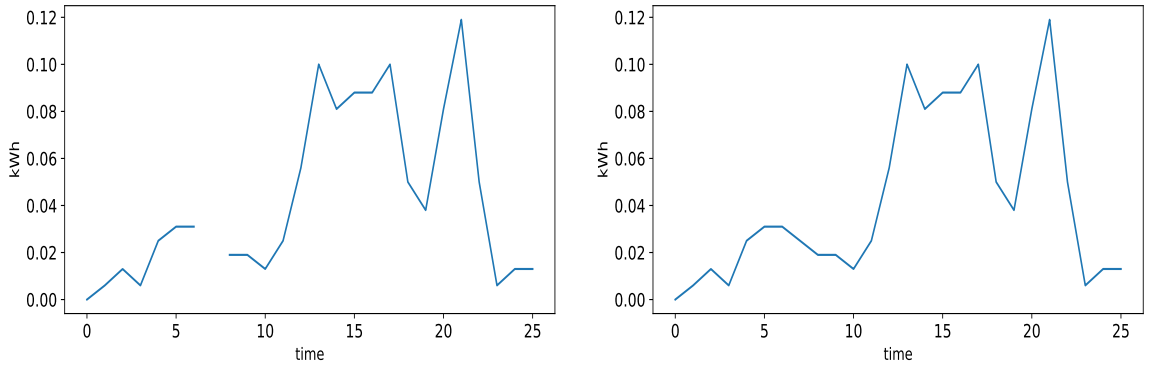


Figure 6.4.: Day with a missing observation before and after linear interpolation.

For time series with a wide range of missing values, we insert the data points of a day with a similar date from the preceding or following years. For example, an unusable day on the 23.7.2011 might be replaced by the 23.7.2012 or the 23.7.2011. In this way, we preserve seasonal weather characteristics that the day is exposed to. In general, we had to replace 15 unusable days on average for every time series.

6.2.2. Weather-based Approach

The generation of photovoltaic systems has a close relationship to weather conditions, such as the temperature, humidity, solar irradiance, hourly solar angle, to name a few. E.g., a time series that is made up of power output sequences on only sunny days is going to be different from a time series made up of the power output sequences generated in rainy days. To the best of our knowledge, time series datasets containing weather information have been widely used in forecasting, where the efficient forecasting features based on weather conditions are extracted and used to increase the predictive accuracy of the forecasting models. In our weather-based approach, we introduce new usage of weather data by using information about weather situations and corresponding power production to generate new time series datasets.

The basic idea of the weather-based method is as follows: based on the input time series and daily weather data for the same period, we filter the input time series according to

certain weather conditions and use the filtered time series as a new one. For example, we can characterize each day in the solar power time series as rainy or sunny as seen in Figure 6.5. We can then create a new time series by only keeping the rainy days and removing the sunny days or vice versa. With our weather-based approach, we intend to generate new time series datasets, in which the days inside it share the similar weather conditions but their day-to-day order is modified. This will in turn generate different time series with different characteristics.

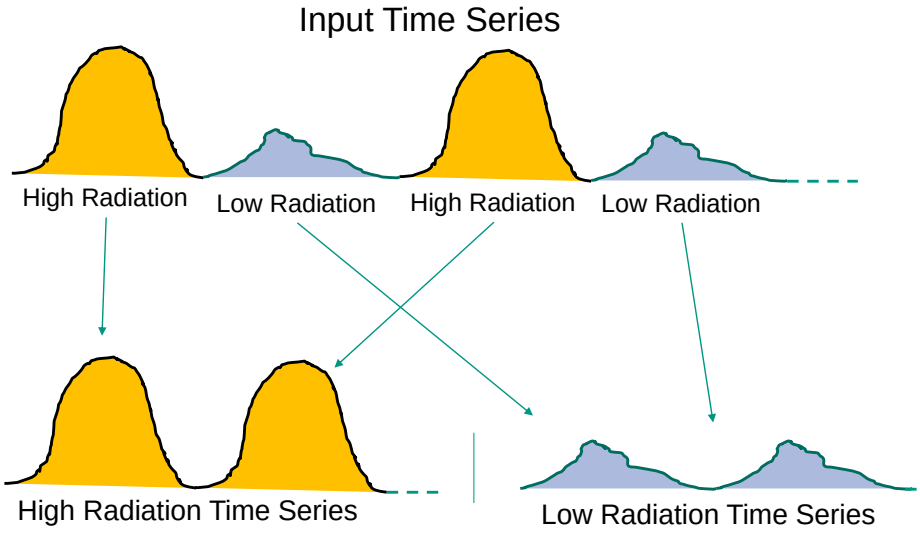


Figure 6.5.: Filtering input time series datasets based on weather conditions.

The proposed weather-based approach can be applied independently to each input time series dataset. We classify each day in the input time series based on the available weather data, which describe the weather conditions of that day. E.g., a day may be labeled as being sunny, or very hot, or both. After all the days have been classified, we chain all the days in each class separately in chronological order to create a new time series. As a result and for each input time series dataset, we can create a new time series for every weather class.

The weather data used should correspond to the power production data as best as possible. Because the production data reflect productions of households with PV in Australia, corresponding weather data from the "Bureau of Meteorology of the Australian Government" is used. Further description of this time series dataset is provided in Section 2.4.3.

It contains the daily weather information for the same location of our input time series datasets presented in Section 6.2.1. Such information includes evapotranspiration, maximum and minimum relative humidity, maximum and minimum temperature, rain and solar radiation, to name a few. From these weather observations, we derived 14 classes for characterizing the weather situation of a day as listed in Table 6.1. In this table, each class is briefly described by a short description of which weather conditions need to be present in order for a day to be assigned to the respective class.

www.bom.gov.au

Table 6.1.: Weather classes defined in the weather-based approach.

Weather Class	Description of Weather Class
High/low radiation	Higher/lower radiation than the median
Fluctuating/steady temperature	Difference between maximum and minimum temperature of a day is higher/lower than the median difference
High/low temperature	Both maximum and minimum temperature are above/belowtheir respective median
High/low evapotranspiration	Evapotranspiration is higher/lower than the median
Rainy	Precipitation higher than 0
Not_rainy	Precipitation equal to 0
Fluctuating/steady humidity	Difference between maximum and minimum humidity of a day is higher/lower than the mean difference
High/low humidity	Both maximum and minimum humidity are above/below their respective median

Given an input time series dataset, we filter it based on the aforementioned classes and consequently generate new time series. As only 14 weather classes are taken into consideration in filtering the input time series dataset, we can generate 14 new time series for each input one. As mentioned in Section 6.2.1, 238 input time series datasets are resulting at the end of the preprocessing procedure. Consequently, $14 * 238 = 3332$ new time series datasets are generated in total by our weather-based approach.

Because certain days are removed from the input time series datasets to arrive at new one based on weather conditions, the new generated time series datasets will contain less observations than the input time series datasets, from which they are derived. E.g., the number of observations in the new generated time series datasets range between 8736 and 16432 observations, compared to the 28496 observations present in each input time series dataset. The shortest newly generated time series dataset consists of days with high humidity, whereas the longest time series dataset was made up of days classified as not_rainy. Although this approach for generating new time series datasets based on weather information is quite simple, it is quite effective as the evaluation described later in this chapter will show.

The approach is limited to that it can only be applied to generate new time series datasets from power generation ones. If the input time series datasets contain load values, then this approach cannot be applied to create new meaningful load time series, as the load data are not affected by weather but instead by human behavior.

6.2.3. Aggregation-Based Approach

The main goal behind this approach is to aggregate the input time series datasets using different aggregation levels in order to generate new time series datasets with different characteristics. This makes sense because energy can be summed up over a larger time interval to form a new energy time series data set with lower time resolution but having the energy production summed up over the larger interval. These datasets will then be

transformed into meta examples and merged in the learning process of meta learner to increase its accuracy as will be seen in Section 6.3.

For each input time series dataset resulting from the preprocessing process in Section 6.2.1 and during the aggregation process, multiple data points are added up in a given interval to create new aggregated observations as seen in Figure 6.6. We will refer to the number of data points that are aggregated to a single data point in the aggregation process as an aggregation level. Only period S explained before in Section 6.2.1 is taken into consideration in each input time series dataset. In this way, we ignore the large night time gap between the last observation of the previous and the first observation of the next day. The time interval between two data points defines the granularity of the time series dataset. E.g., the granularity is half-hourly for half-hourly observations of our input time series datasets. Moreover, when each day contains only one value, the time series has daily granularity.

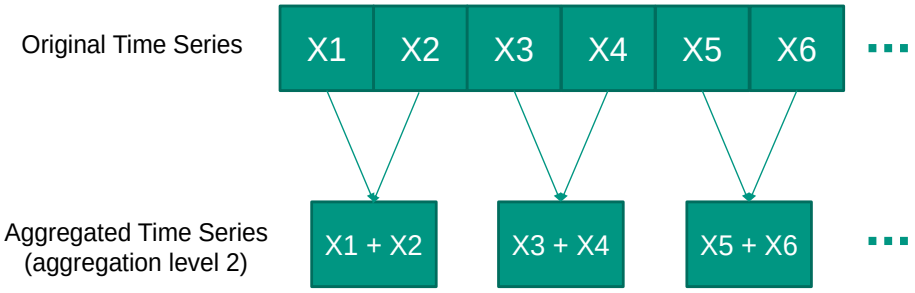


Figure 6.6.: Methodology of the aggregation-based Approach for generating new time series datasets.

Given n observations per day, the aggregation levels can range between 2 and n. As one-day-ahead forecasting is considered in this work as a forecasting scenario as will be seen in Section 6.3.1, the aggregation levels larger than one day will not be taken into consideration, since they would result in data points containing observations from multiple days. Furthermore, depending on the aggregation level, the forecasting horizon for a time series has to change. If the algorithms were using the original forecasting horizon of n for a time series which has daily granularity, the algorithm would end up forecasting n days ahead, instead of doing a one-day-ahead forecast. Thus for an aggregation level of n, i.e. daily granularity, the forecasting horizon will be 1, whereas for an aggregation level of $\frac{n}{2}$ the forecasting horizon is 2 and so on. More details about the forecasting scenario for which our meta learning problem is formulated, are provided later in Section 6.3.1.

In contrast to the aforementioned weather-based approach which can only be applied to generate new time series datasets if the input contains power generation values, the advantage of this approach lies in its applicability in the context of load and generation datasets. Driven by the aforementioned methodology of the aggregation-based approach, we aggregate each input time series dataset with 8 aggregation levels, namely 2, 3, 4, 5, 6, 8, 13, 26. Consequently, with 8 aggregation levels and 238 original time series, we are able to generate $8 * 238 = 1904$ new aggregated time series.

The lengths of the newly created time series range from 1096 for the largest aggregation level up to 14248 data points for the smallest aggregation level. The impact of the different aggregation levels on the periodicity and granularity of the generated time series can be seen by looking at Figure 6.7. Each graph in the figure shows 50 data points of the new time series which were generated with the aggregation-based method.

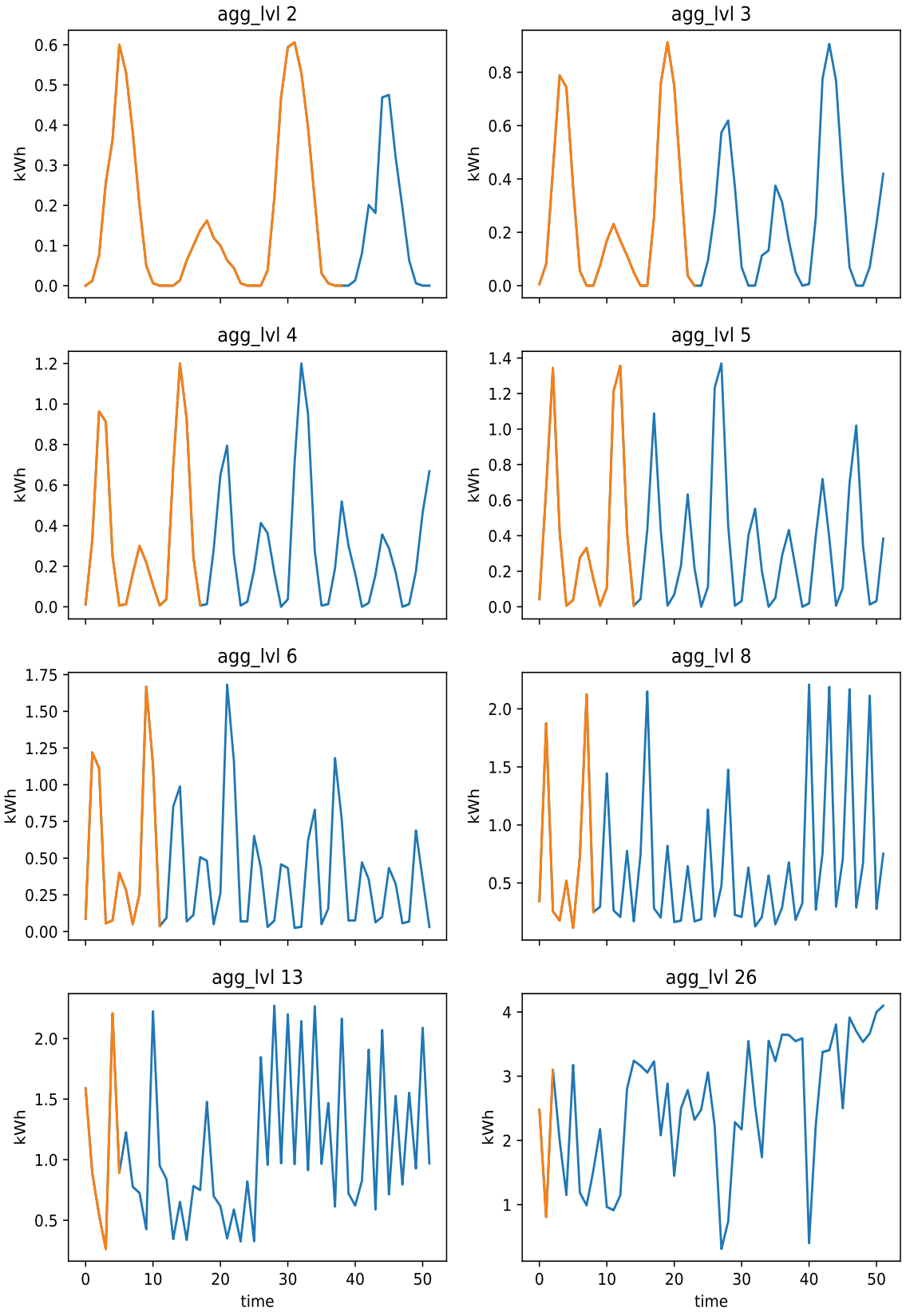


Figure 6.7.: Examples of the time series generated by the aggregation-based approach.

6.2.4. Model-Based Approach

To evaluate the efficiency of the aforementioned weather- and aggregation-based approach, we need to compare them with model-based approaches existing in the literature [156] and used to generate new time series datasets by simulating new ones as seen in Figure 6.8. X and Y represent the input features and forecasted values respectively.

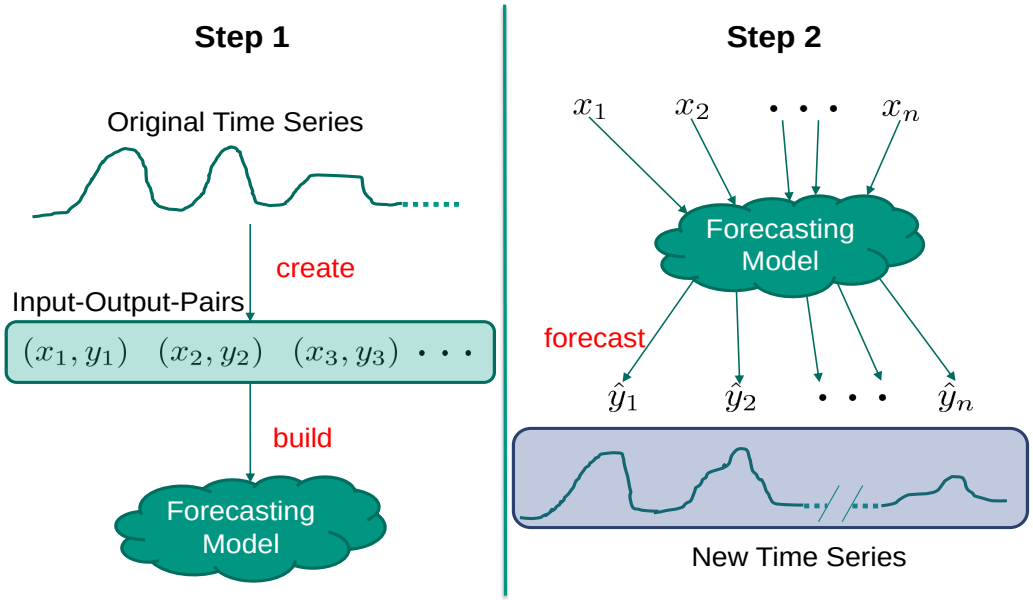


Figure 6.8.: Methodology of the model-based approach.

The model-based approach incorporates two main steps. In the first one, the forecasting model is trained on the original training time series datasets. However, more details about the training performed in this forecasting scenario is presented in Section 6.3.1. In the second step, the trained model is used to generate new data points based on the model's learned parameters where the new generating data points correspond to the forecasted values. Precisely, the forecasting model is used to perform forecasts based on the constructed input vectors. By appending these forecasts together, we can create a new time series dataset. To achieve that, 4 ML algorithms are used, namely Neural Network (NN), k-Nearest Neighbor (KNN), Decision Tree (DT) and Random Forest (RF). 8 different setups shown in Table 6.2 are employed to build the models. While 3 models are developed for each of NN and KNN, 1 model is developed for each of DT and RF. We used keras framework version 2.3.1 to implement neural network models. The classes DecisionTreeRegressor and RandomForestRegressor in scikit-learn are used to build DT and RF models respectively.

In this approach, we used 238 input time series datasets presented in Section 6.2.1 to simulate new ones. Each input time series consists of half-hourly observations over the period of 3 years. Given an input time series, one model is built for each algorithm. The models are trained to forecast one-day-ahead based on an input vector. The input vectors contain information about the day preceding the forecast. Further details about the feature

Table 6.2.: Hyperparameters used in building model in the model-based approach.

Model Name	Hyperparameters
NN	It contains 2 hidden layers and 20 node in each one of them
NN_MODIFIED1	It contains 1 hidden layers and 50 node in each one of them
NN_MODIFIED2	It contains 3 hidden layers and 50 node in each one of them
KNN	K=3
KNN_MODIFIED1	K=5
KNN_MODIFIED2	K=10
Decision Tree	A grid search is applied for the maximum tree depth parameter in the range from 5 to 15
Random Forest	200 regression trees

engineering process and the values of the input vectors are introduced in Section 6.3.1, where the same input features are used here. Each day of the original input time series consists of 26 observations, leading to forecast 26 values in our one-day ahead forecast.

After building the models, the same training input vectors were fed to the models generating one-day-ahead forecasts for each of the input vectors. For each model, the forecasts were then appended to create a new time series. For each algorithm and each input time series, a new time series was generated. Consequently, we were able to generate 1904 new time series with the same length and granularity of the original ones. The efficient preprocessing procedure presented in Section 6.2.1 including upward outliers and filling missing values is applied on the resulting new generated time series. Figures 6.9 and 6.10 show examples of the new generated time series datasets. While the whole three year period of the generated time series is shown on the left, an excerpt of three days of that time series is depicted in the right figures.

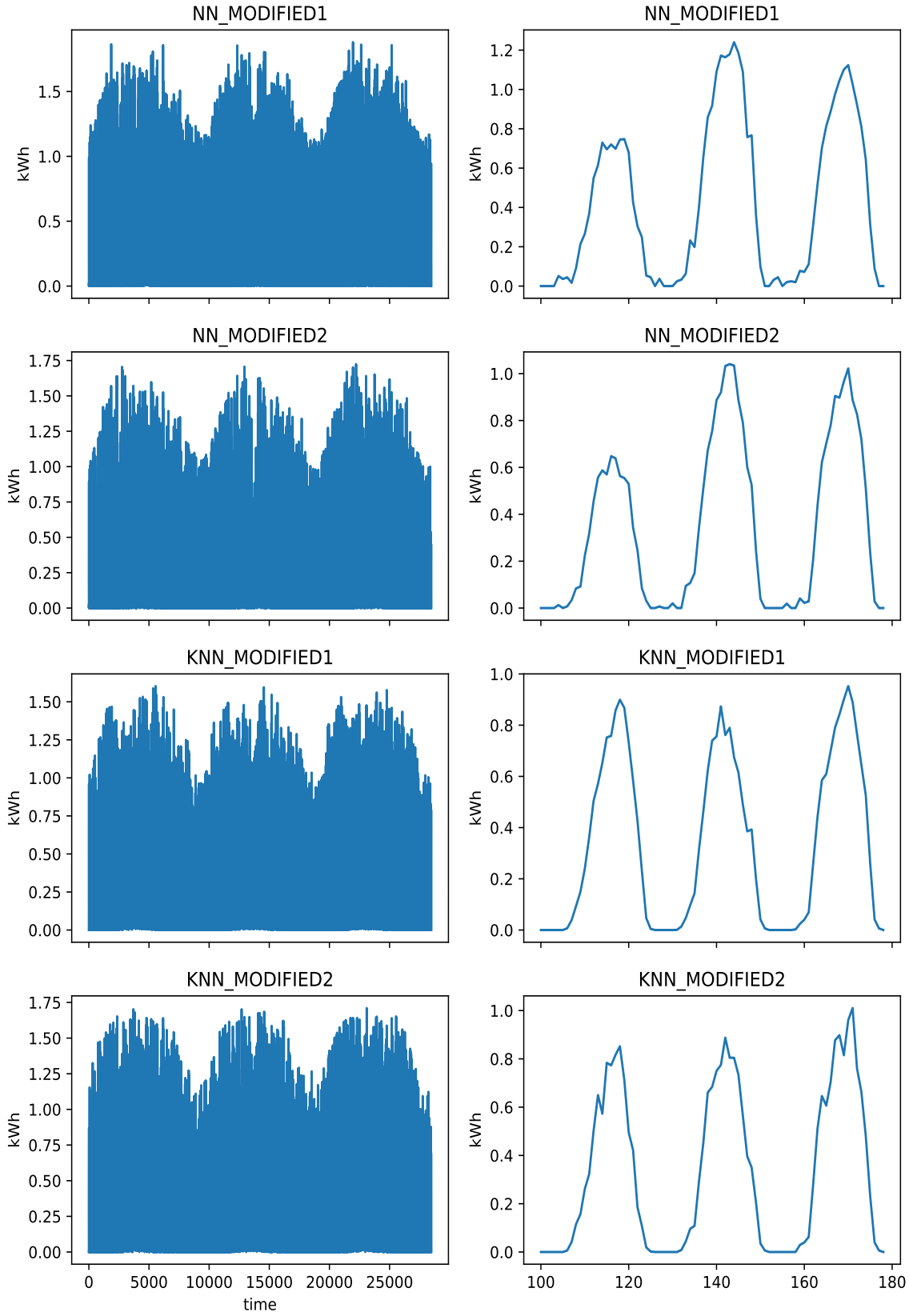


Figure 6.9.: Examples of the time series generated by the model-based approach.

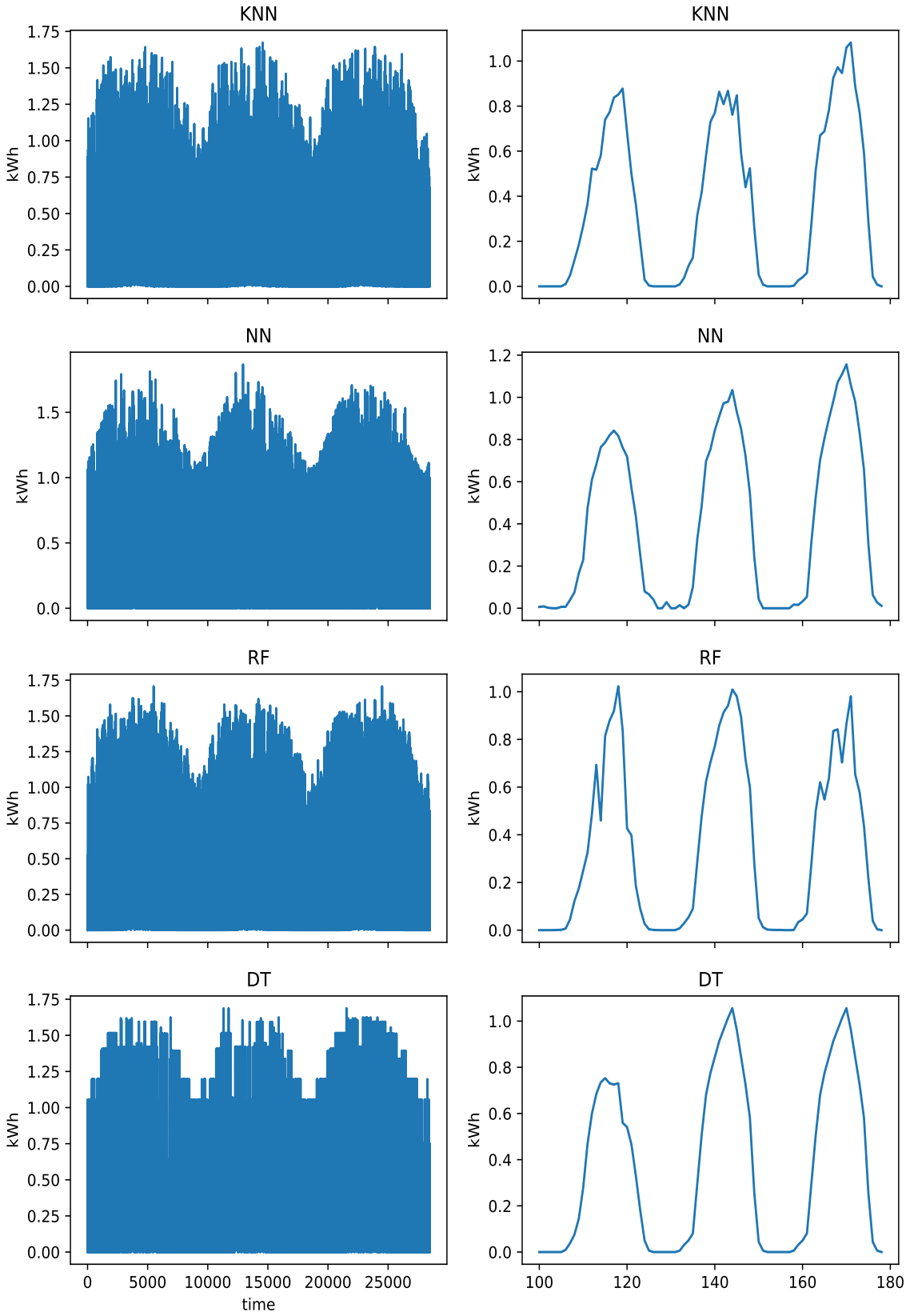


Figure 6.10.: Examples of time series generated by the model-based approach.

6.3. Evaluation

As mentioned before, our meta learning approach is formulated as a multi-class classification problem in which the number of available examples plays an essential role. To generate new examples, we proposed two approaches, namely a weather-based and an aggregation-based approach. Those examples are then used to generate the related meta examples by extracting the meta features and assigning the label of the best forecasting model for them.

The key focus of the evaluation is to investigate the effect of increasing the number of meta examples on the predictive performance of meta learners. We compare the effects of the new meta examples generated by our approaches with the effect of the meta examples generated by a model-based approach existing in the literature. We start by presenting the meta learning use case scenario including the extraction of the different groups of meta features to characterize the input energy time series datasets. After that, an in-depth analysis study of the effect of the new meta examples is performed in two points of view. While the original representation of meta features is used in the first one, we utilize unsupervised deep learning to encode the extracted meta features by autoencoder in the second point of view (as shown in Chapter 5).

We apply a meta learning approach on the one-day ahead forecasting for solar power generation scenario presented later in Section 6.3.1. We set the predictive performance of the meta learner when it learned on the original input datasets as a baseline for our evaluation. Then, we investigate how the predictive performance of the meta learner changes, when it learns on extended training datasets. To this end, we start by transforming the newly generated time series into meta examples, where each meta example consists of the calculated meta features as predictors and the best forecasting model as a label. The better the used meta features characterize the behavior of time series datasets, the more accurate the selection of the best algorithm by a meta learner will be. Therefore, Time series, Statistical, Information-theoretic and Descriptive Statistics Time-based Meta Features (DSTMF) presented in Table 6.3 are extracted in our experiments.

As we see in this table, DSTMF for solar power generation time series datasets does not include the same aggregation levels as it were in the case of consumption. E.g., it doesn't make sense to include weekends and workday aggregation levels. This is due to the fact that the electricity power generation of photovoltage power systems is only related to weather conditions and not affected by human behavior. In this group of experiments, we used R packages to extract a total of 67 meta features presented in Table 6.3. 44 time series meta features are extracted using tsfeatures in R. In Chapter 5, we found that the Time Series and DSTMF meta features are the best meta features for characterizing energy time series datasets. Therefore, we focus in our evaluation here on both of them and extract only 1 meta feature for each of Statistical and Information-theoretic category.

Table 6.3.: Different groups of the extracted meta features.

Meta Feature Category	Meta Features
Time Series	Lumpiness, stability, flat_spots, unitroot_kpss, unitroot_pp, x_acf1, x_acf10, diff1_acf1, diff1_acf10, diff2_acf1, diff2_acf10, x_pacf5, diff1x_pacf5, diff2x_pacf5, alpha, beta, arch_acf, garch_acf, arch_r2, garch_r2, nonlinearity, ARCH.LM, ac_9, firstzero_ac, trev_num, motiftwo_entro3, walker_propcross, std1st_der, spreadrandomlocal_ meantaul, histogram_mode, outlierinclude_mdrmd, fluctanal_prop_r1, trend, spike, linearity, curvature, e_acf1, e_acf10, max_kl_shift, max_level_shift, max_var_shift, mean_crossing_points
Statistical	Var
Information-theoretic	Entropy
DSTMF	orig_max, orig_mean, orig_upper_quartile, orig_lower_quartile, orig_median, daily_max, daily_min, daily_mean, daily_upper_quartile, daily_lower_quartile, daily_median, weekly_max, weekly_min, weekly_mean, weekly_upper_quartile, weekly_lower_quartile, weekly_median, 4-weekly_max, 4-weekly_min, 4-weekly_mean, 4-weekly_upper_quartile, 4-weekly_lower_quartile, 4-weekly_median

6.3.1. Use Case Study: Power Generation Forecasting Scenario

In this section, we introduce the time series forecasting scenario which is used to evaluate our meta learning approach presented in Section 6.2. The proposed forecasting scenario is to forecast the solar power generation for the dataset presented in Section 6.2.1. To enable meta learners to recommend the best forecasting model for energy time series datasets, meta examples are required. In the corresponding meta examples, the label presents the forecasting model that performs best on the corresponding time series. To calculate these labels, many forecasts using different forecasting algorithms were performed in order to arrive at a sensible label for each time series. This process is crucial for the meta learning system, because wrongly classified meta-examples can negatively impact the classification accuracy of the meta learner.

Depending on the use-case, the forecasting horizon can be categorized into different categories, namely short-, very-short, mid- and long-term forecasting. While the long-term forecasting is considered for maintenance related issues, the short-term and very-short-term focuses are most important in the areas of storage control, automatic generation control, unit commitment and the electricity market [169].

In our evaluation scenario presented in this chapter, we will focus on the task of one-day-ahead forecasting of the solar power generation in the form of multi-step forecasts. For all of the time series datasets resulting from preprocessing as explained in Section 6.2.1, a forecasting horizon of 26 is used, because the number of observations in each day is 26 as discussed before. However for the time series datasets generated by the aggregation-based approach, the days can contain less than 26 observations. This means that the concrete forecasting horizon changes depending on the granularity of the time series to always match the one-day-ahead forecasting task.

In contrast to some forecasting approaches that use exogenous weather data like temperature and humidity as features to forecast energy power generation, we follow a univariate, data-driven forecasting approach whereby the forecasting of the power generation is based on historical data without using exogenous features to support forecasting. In general, the question whether linear or non-linear algorithms are more appropriate for time series forecasting, especially univariate time series, has no clear answer and depends on various factors, including the available sample size [31]. Because of this, Cerqueira et al. recommended that the experiments on forecasting should include both types of methods. To this end, both linear and nonlinear forecasting algorithms are included in our forecasting scenarios. On one hand, we chose Neural Network (NN) and K-Nearest Neighbor (KNN) as nonlinear algorithms. On the other hand, we chose Auto regressive Integrated Moving Average (ARIMA) and Error Trend Seasonal (ETS) as linear algorithms.

Forecasts can be divided into multi-step ahead and single-step ahead ones. While only a single value is forecasted in single-step ahead forecasting, multi-step ahead forecasting concerns itself with forecasting multiple future values at once. The number of forecasted values in a multi-step-ahead forecast is called the length of the forecast or the length of the forecasting interval. However, there are three basic forecasting strategies, when it comes to multi-step forecasting [155], namely iterated, direct and Multiple-Input Multiple-Output (MIMO) forecasting.

With the iterated forecasting strategy, a single model performs multiple consecutive single-step forecasting. In each iteration, the value of the single-step forecast is fed back into the model, as if it was the most recent value of the time series. Based on this new given value, the model can then again perform a new single-step forecast. This is repeated until the forecasting horizon is reached. After that, all one-step forecasts are combined to form a multi-step forecast. This strategy has the disadvantage that it is very sensitive to early errors, i.e. the errors which are made at an early stage will propagate to future predictions and make them worse. In this way the errors accumulate more and more, the longer the forecasting horizon becomes.

By using a direct forecasting strategy, also called independent forecasting, a number of models are trained where each of them performs a point forecast with different forecasting horizons. The multi-step forecast is then formed by combining the forecasts of these different models. The errors of the forecasts of this method don't accumulate. However because the models are independent from each other, this strategy can fail to capture dependencies between the predicted values.

For the MIMO forecasting strategy, the models can directly forecast more than one value at once. Such models can be e.g. neural networks, where the number of neurons in the output layer has the exact length of the forecasting interval. This strategy does not have the same disadvantages as the other, but it is less flexible since the same model is used to predict all future values.

For our evaluation instrumenting multi-step forecasting, the NN and KNN models follow the MIMO strategy, whilst ARIMA and ETS follow the iterated forecasting strategy. The KNN forecasting model is implemented with the python library scikit-learn version 0.21.3. ETS and ARIMA models are implemented in R with the package forecast version 8.12. NN models are implemented using keras version 2.3.1. All of these forecasting models except the NN models are made deterministic with a seed for better reproducibility. In the following, a brief explanation of hyperparameter settings of the aforementioned algorithms is presented.

- **Neural Network (NN):** To implement the neural network, we used the python library keras. The concrete network architecture consists of two hidden layers consisting of 20 densely connected neurons. Each of which use the ReLU activation function. The size of the output layer is equal to the length of the forecast horizon. The neurons in the output layer use a linear activation function. MAE is used as the loss function and rmsprop for optimizing the batch learning. We train the neural network for 500 epochs and set the batch equal to the length of the training set. We also use early stopping with patience 20.
- **K-Nearest Neighbor (KNN):** This algorithm was implemented using the scikit-learn class KNeighborsRegressor. The parameter k is set to 5, i.e. 5 neighbors are used with a uniform weighting.
- **Autoregressive Integrated Moving Average (ARIMA):** We use the auto.arima function from the R forecast package with the following settings: stepwise=TRUE, approximation=TRUE, allowmean=FALSE, truncate=5000 and method=CSS.
- **Error Trend Seasonal (ETS):** To implement the ETS algorithm, the stlm function from the R forecast package with the ets method was used. The seasonal component is removed before the ets function is applied and afterwards added back to the predictions of the ETS model. The ets function automatically tries differently parametrized ETS models and selects the best one using the Akaike Information Criterion (AIC). We need to seasonally adjust the time series, because the ets method can only handle data with periodicity smaller than 25 and our original half-hourly time series have daily periods of 26 data points.

When considering how to build the models and which input features to feed the forecasting algorithms, we differentiated between the linear models, namely ARIMA and ETS and the nonlinear ones, namely KNN and NN. For the linear models, the values in the training set are standardized. Then, based on this normalized time series, the ARIMA and ETS algorithms can build the forecasting model. After the model has made a forecast, the predicted values are transformed back to the original scale using the variance and mean

from the training set. By using the mean and variance of the training set, we prevent data leakage from the test set.

For nonlinear models, the model building and feature engineering process is more complex. While we don't need to explicitly construct input-output pairs for the linear models, we have to do so for the nonlinear ones. The input vectors can contain information about historical data points and timestamps, while the training output vector is the sequence of actual values $(y_t, ..., y_{t+h})$ of the day that is to be forecast. To construct the input vectors, we extract various time series features to support this learning process. Let y_i^s denote the value y_i that has been normalized based on the training set, then we extracted the following features:

- The 26 lagged, normalized data points $y_{t-1}^s, ..., y_{t-26}^s$.
- The first-order differences $d_{t-1}, ..., d_{t-25}$ of the normalized data points $y_{t-1}^s, ..., y_{t-26}^s$ where $d_i = y_i^s - y_{i-1}^s$ making up 25 features.
- The encoded values of month and day of the month of the day that is to be predicted. We first combine month and day to a value representing the ordinal day in the year and we then encode this value with a combination of a sine and a cosine function. The ordinal day o with periodicity p is encoded with $f_s(v) = \sin(\frac{2\pi o}{p})$ and $f_c(v) = \cos(\frac{2\pi o}{p})$. This results in 2 more features.

As a result, we extracted 53 features to formulate the input vector in the nonlinear forecasting models. The observed values for the next day, which are to be predicted, make up the output vector $(y_t, ..., y_{t+h})$, where h is the forecast horizon. In nonlinear forecasting models, we don't need to transform the output values like we did with the linear algorithms. After creating the input-output vectors, the model can be trained to capture the relationship between input and output vectors. For the forecast, an additional input vector which corresponds to the test day is constructed and supplied to the model, which transforms it into an output vector, i.e. the one-day ahead forecast.

When trying to estimate the out-of-sample forecasting performance of a model on a time series, multiple out-of-sample forecasts should be performed to reduce estimation variance. For classification tasks, cross-validation is often used to better approximate the out-of-sample error as opposed to having a fixed test set. For time series forecasting, the ordinary cross-validation can't be used, because it doesn't preserve the temporal structure of the dataset and can cause data leakage, i.e. models could be able to make predictions based on future values that aren't available at the time of the forecast. To avoid this problem, a variation of cross validation can be used known as time series cross validation or nested cross validation [15].

The main idea behind nested cross validation is to split the time series at some time tp into training and test sets. Then a rolling forecast is performed on the test set in which we train the model anew at every step of the iteration, i.e. before each new forecast. It allows the models to use all past observations prior to the forecasting time for training as seen in Figure 6.11. However, it can be computationally expensive, if the model takes a long time to train. As we can see in this figure, each row represents one iteration. The red dots

represent forecasts and the blue dots are used for training. The average forecast error of all these forecasts can be used to better approximate the out-of-sample forecasting error of the final model.

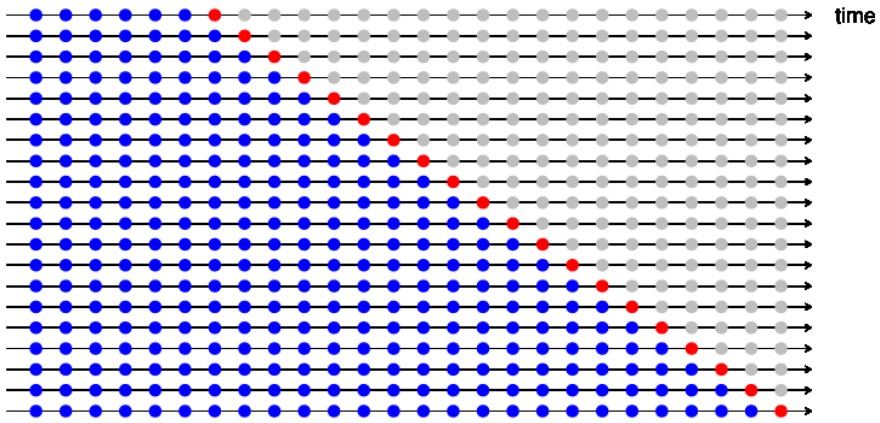


Figure 6.11.: Methodology of nested time series cross validation [15].

Generally, there is a trade-off when choosing how many one-day-ahead forecasts should be performed on a time series to measure algorithm performance on the time series. Increasing the number of forecasts yields more reliable results in terms of assessing algorithm performance. However, since for every forecast a new model is built, the computational cost of doing so also increases with every forecast. Considering the constraints of the available computational power and time, we chose to limit the extent of testing by restricting the algorithm to 28 one-day-ahead forecasts per time series. 28 days lead to 728 forecasted values or 4 weeks worth of forecasts for the half-hourly granularity of the original dataset.

To measure the error of a forecast, we can't use percentage-based error measurements, because the time series contains zero values. Thus, we decided to use the MASE with periodicity 1 to measure the error of each multi-step one-day-ahead forecast. By using the MASE, we don't weigh larger errors more heavily than smaller errors, as it is done e.g. by the MSE, which squares the errors. The average of these forecast errors, i.e. the mean MASE, then represents the algorithm performance on the time series. Driven by the aforementioned forecasting strategy, we perform our multi-step one-day-ahead forecasting on the original and new generated time series datasets resulting in a total of 7475 time series datasets to be forecasted. This part of the work was very computing-intensive and couldn't be carried out on a simple home computer. To face this challenge, we carried out the computations using Kubernetes and Docker images on a computation cluster. We used 10 pods with each 3 cpu and 10 gigabyte memory to perform the calculations in parallel.

Figures 6.12 and 6.13 show the mean MASE forecasting error of the ARIMA and NN forecasting models. The obtained error is used to determine the corresponding labels in the final knowledge matrix, where the algorithm that performs best on a time series dataset is set a label for it in the final knowledge matrix.

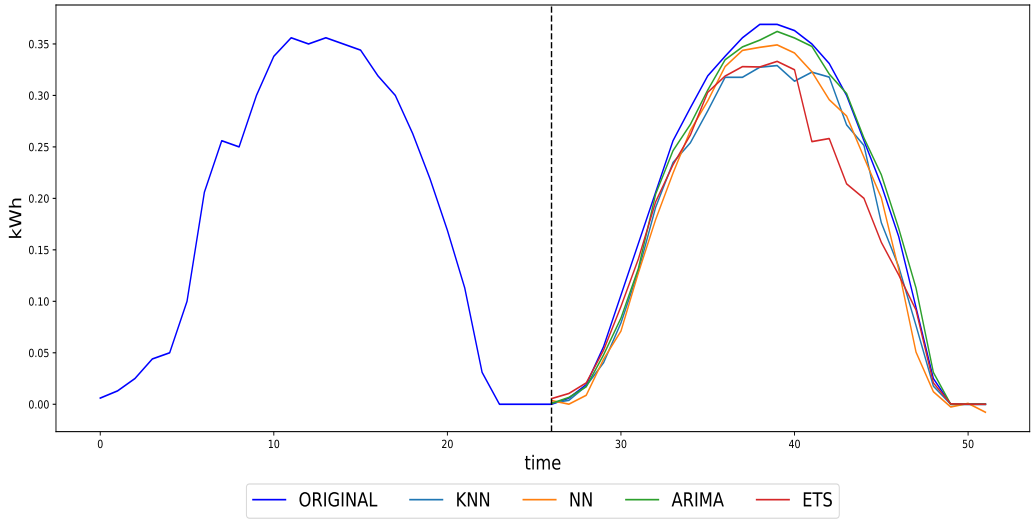


Figure 6.12.: One-day ahead forecasts, where ARIMA performed best.

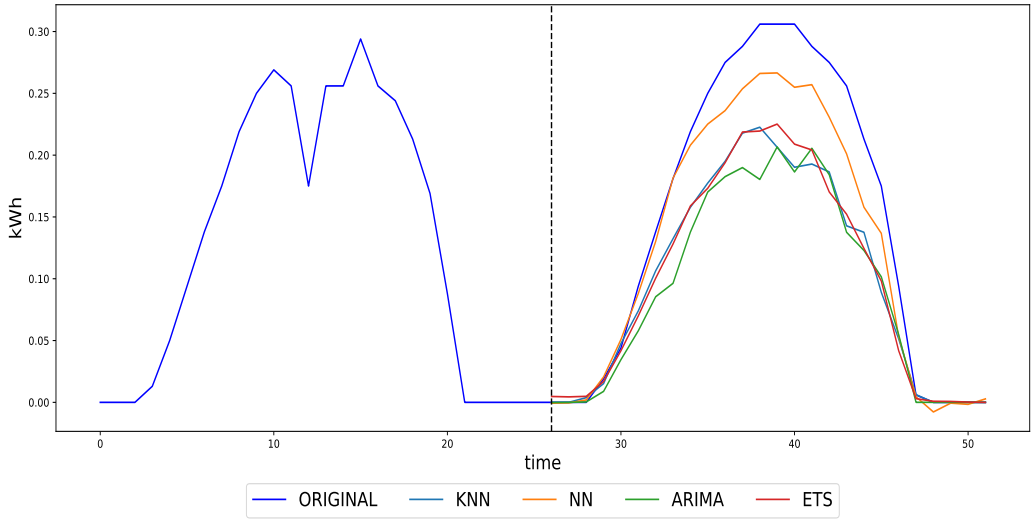


Figure 6.13.: One-day ahead forecasts, where NN performed best.

6.3.2. Original Representation of Meta Features

Due to the large number of extracted meta features, the curse of dimensionality problem negatively affected the predictive performance of the meta learning classification model leading the model to overfill the training data. Moreover, a large number of meta features are originally highly correlated with each other. To address these problems, we applied an efficient feature selection procedure to reduce the number of meta features and remove the highly correlated ones. E.g., if we have two highly correlated features, then one of them is removed. Using the Pearson correlation coefficient as a correlation measure, we performed a correlation analysis on the aforementioned set of meta features and remove the meta features that are highly correlated. We successively removed the meta features that have a correlation coefficient of 80% or higher to another meta feature. As a result of features

selection, only 25 meta features are used in this group of experiments for characterizing energy time series datasets.

As mentioned before, the knowledge matrix on which the meta learner is trained consists of a set of meta examples, whereby each example represents a time series. While some meta examples represent the original input time series, the other ones represent time series that are generated by our new proposed generation approaches. To differentiate between these meta examples, we refer to them as aggregation-based, weather-based, model-based and original meta examples. Artificial Neural Network (ANN) with a single hidden layer of 40 neurons is used as a meta learner to serve the main goal of meta learning in recommending the best power generation forecasting model. In the current meta learning use case, the meta learning classification model will classify the energy time series datasets in testing in one of four forecasting models, namely NN, KNN, ETS, ARIMA presented in Section 6.3.1.

In this group of experiments, we used the 238 original time series datasets resulting from the preprocessing procedure in Section 6.2.1. For those datasets, we build the relevant meta examples as explained before. To generalize the result, we repeated our experiments 100 times and the mean results were used as final result values. In each iteration, we split the original meta examples into training and testing sets randomly. The size of the testing set is 24 meta examples equaling 10% of the original meta examples.

The remaining 214 meta examples will build the original training datasets and are extended later to contain other meta examples resulting from the other categories, namely weather-based, aggregation-based and model-based. In each iteration, we extend the original training dataset with the newly generated meta examples in such a way that the original label distribution is preserved. This means that the extended datasets have the same label distribution as the original training set. To this end, performance differences will not be affected by the distribution of labels in the extended training datasets.

To precisely measure the effect of the new generated meta examples on the predictive performance of meta learner, we extend the training dataset on which the meta learner is trained according to 7 different sizes. While the original training dataset always contains 214 meta examples, the extending training dataset will contain 321, 428, 642, 856, 1070, 1284 and 1498 meta examples. These sizes correspond to 1.5, 2, 3, 4, 5, 6 and 7 times the size of the original training set size. Table 6.4 shows the different extending scenarios performed in our experiments.

We created 4 extension scenarios for each of the aforementioned sizes of the training datasets meaning that we have 28 extended training sets in each iteration and a new meta learning model is built for each of those. After that, we normalize the meta features in the new extending training sets into a range between 0 and 1 to improve the learning process of the ANN meta learner. This is due to the fact that the normalization will lead the classification model to find the global minima more quickly. For stable evaluation, we also normalize the meta features in the testing set.

Table 6.4.: Different extending scenarios of the training dataset.

Extending Scenario	Description
Original + Aggregation-based	Extending the original training set with new meta examples resulting from the aggregation-based approach
Original + Weather-based	Extending the original training set with new meta examples resulting from the weather-based approach
Original + Model-based	Extending the original training set with new meta examples resulting from the model-based approach
Original + Aggregation-based + Weather-based	Extending the original training set with new meta examples resulting from a combination of aggregation-based and weather-based

In each iteration, the meta learner is trained on the original and 28 extended training datasets. we measure the relative classification accuracy improvement on the test set for each extended training according to Equation 6.1.

$$Acc = \frac{Acc_{ext} - Acc_{orig}}{Acc_{orig}} \tag{6.1}$$

Let Acc_x be the number of correctly predicted labels in the test set divided by the number of meta examples in the test set for dataset x. Consequently Acc_{orig} is the classification accuracy of the meta learner when it learns on the original training set, and Acc_{ext} the classification accuracy of the meta learner when it learns on some extended training set.

We performed 100 iterations, where in each iteration the relative performance improvement is recorded for the extended dataset, which differ by the amount and the category of the meta examples that the original training set was enhanced with. Figures 6.14, 6.15, 6.16, and 6.17 show the mean relative accuracy improvements achieved by the different extension scenarios. The mean relative accuracy improvement of the meta learner, when the original training set is extended using meta examples resulting from the aggregation-based approach, is shown in Figure 6.14 for different training set sizes. As we can see in this figure, adding aggregation-based meta examples increases the meta learner's performance consistently up to a training set size of 856, where a relative performance improvement of 15.60% is achieved. The best mean relative improvement of 16.77% is achieved by 1284 meta examples that correspond to extending the original training dataset with 6 times more meta examples.

In Figure 6.15, the mean relative improvement achieved by extending the original training set with the new weather-based meta examples is presented. As we can see, the relative predictive performance of ANN meta learner increases continuously until a training set size of 1070 meta examples to achieve an improvement of 12.91%. While the best relative improvement of 15.37% is obtained for the largest number of available meta examples in training, the meta learner is able to assign the best forecasting model for the energy

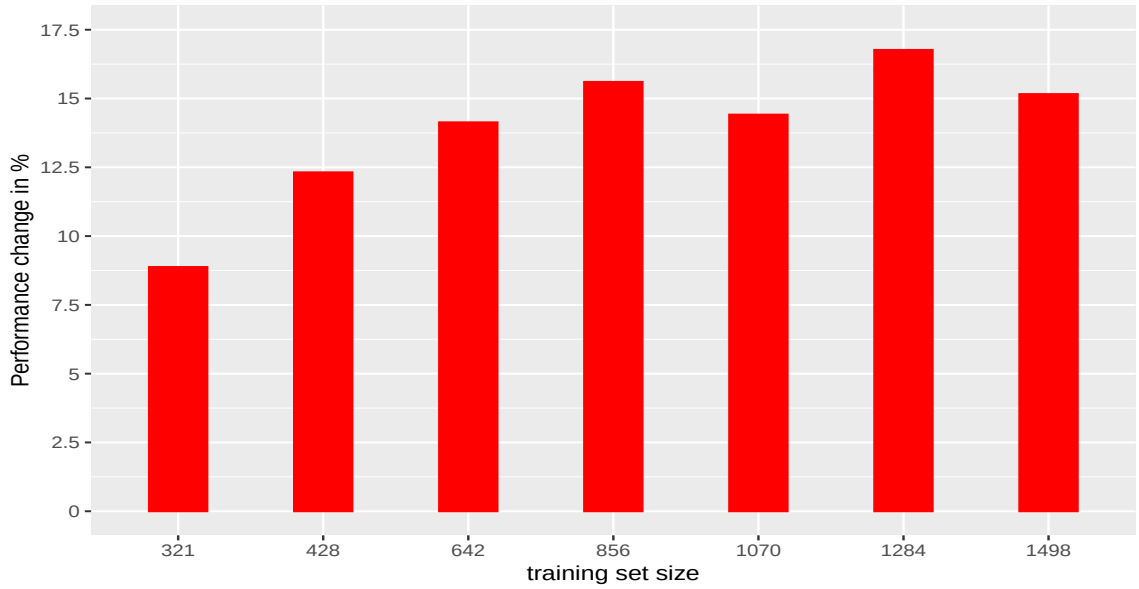


Figure 6.14.: The mean relative accuracy improvement of the meta learner when extending original with aggregation-based datasets.

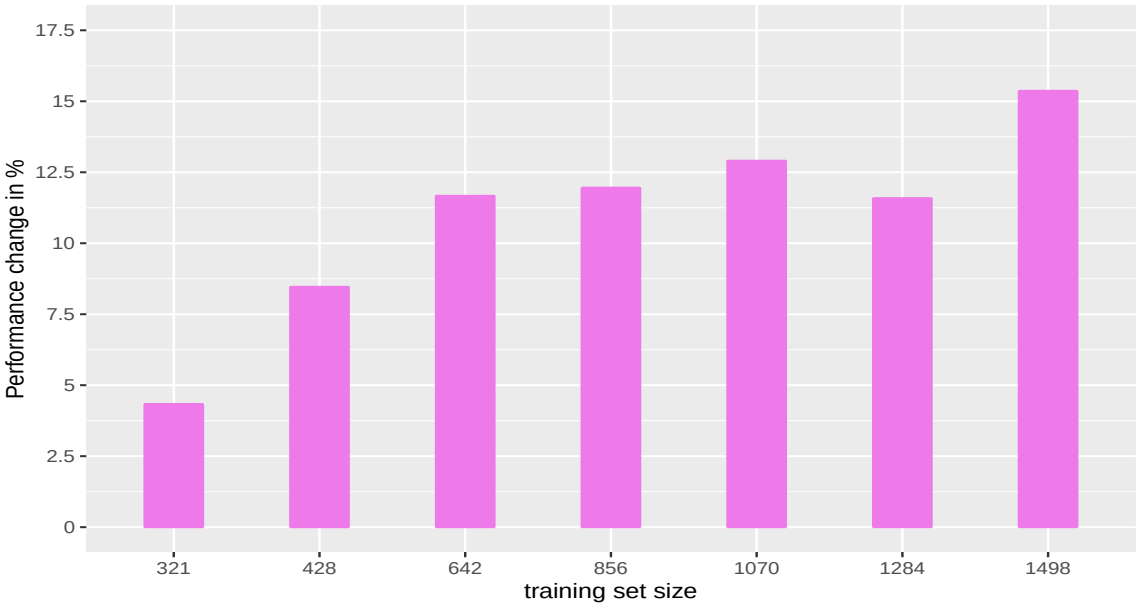


Figure 6.15.: The mean relative accuracy improvement of the meta learner when extending original with weather-based datasets.

time series datasets in testing with a relative improvement of 12.91%, 11.96% and 11.67% at training set sizes of 1070, 856 and 642 respectively.

Figure 6.16 shows the results related to the model-based approach. As seen in this figure, the mean relative improvement in the performance increases continuously until having 856 meta examples where the best relative performance improvement of 14.67% is achieved. However, extending the training set with more meta examples resulting from the model-

based approach seems to have adverse effects on the predictive performance of the meta learner. It is noticed in this figure that the performance improvement drops to 13.02% and 13.14% for training set sizes of 1070 and 1284 respectively. For the largest extension of 1498 meta examples, the performance improvement drops to 11.74%, which is the second worst result for the model-based dataset.

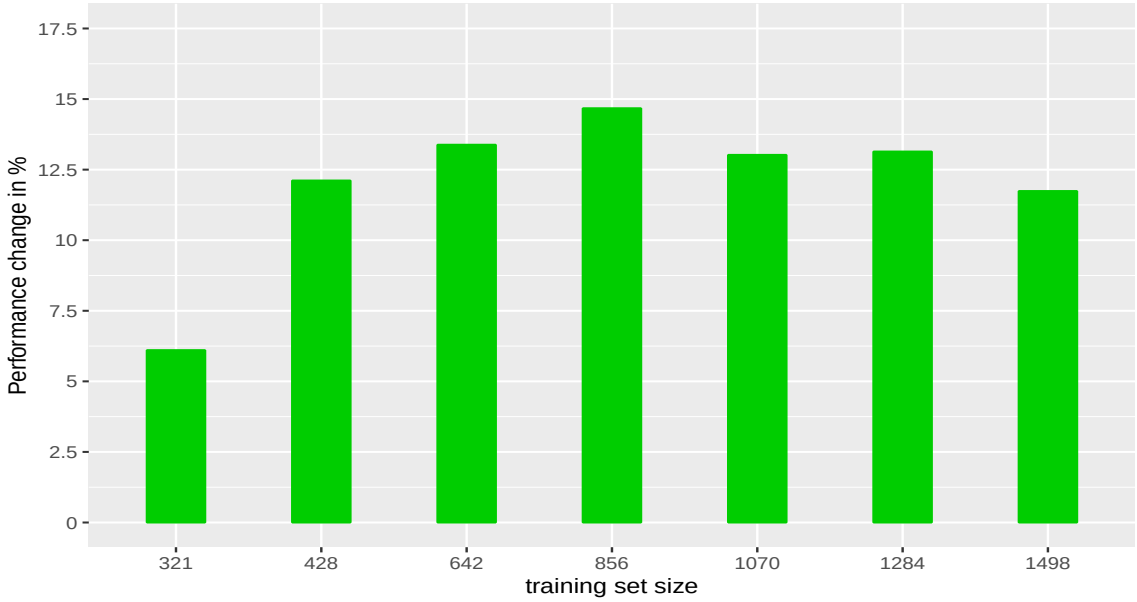


Figure 6.16.: The mean relative accuracy improvement of the meta learner when extending original with model-based datasets.

To further discover the effect of new meta examples on the predictive performance of the meta learning classification model, we extend the original training set with a combination of both weather- and aggregation-based meta features. After that, the predictive accuracy of the meta learner is measured and the relative improvement is calculated according to Equation (6.1) and presented in Figure 6.17. As seen in this figure, the highest mean relative improvement of 14.56% is obtained for a training set size of 1070 meta examples. Extending the training set further did not yield better results where 14% and 14.02% mean relative improvements are achieved for training set sizes of 1284 and 1498 respectively.

For fair evaluation, it is of great importance to compare our two proposed approaches for generating new meta examples with the model-based approach existing in the literature as seen in Table 6.5. As seen in this table, all of the proposed approaches to generate new meta examples enhanced the predictive performance of the meta learning classification model. Aggregation-based approach achieved a mean improvement of 8.75% up to 16.77%. A mean improvement of 4.8% up to 15.37% is obtained when extending the original training set with meta examples resulting from the weather-based approach. The reason for that lies in the fact that the additional examples in the training set increases the diversity and improves the generalization capabilities of the meta learner. Regarding the number of meta examples, the best results are obtained for 1284, 1498, 856 and 1070 for aggregation-based, weather-based, model-based and a combination of weather and aggregation-based respectively.

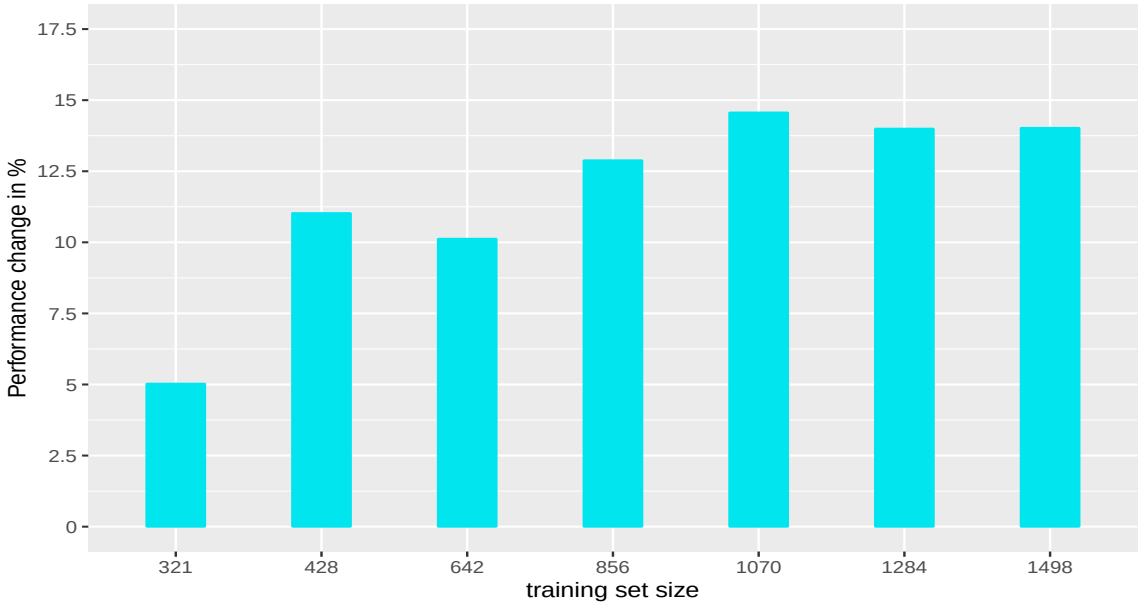


Figure 6.17.: The mean relative accuracy improvement of the meta learner when extending original with combination of aggregation- and weather-based datasets.

Interesting is that the aggregation-based meta examples achieve the best improvement in 5 out of the 7 training set sizes. For these 5 training set sizes, the aggregation-based approach outperforms the next best approach by 2.64%, 0.21%, 0.75%, 0.93% and 2.77% for a 321, 428, 642, 856 and 1284 training set size respectively. For the other 2 training set sizes, the aggregation-based approach is outperformed by the aggregation+weather and the weather-based approaches by only 0.15 and 0.21 percentage points, respectively. However and compared to the model-based approach existing in the literature, the aggregation-based meta examples produced better results than the model-based ones for every training set size as illustrated in Table 6.5.

The main reason for this phenomenon lies in the variance of features in each of the new generated meta examples. In multi-class classification problems, the variance of a feature determines how much it is impacting the response variable. If the variance is high, it implies there is a large impact of this feature on response and vice-versa. To this end, we used R packages to analyse the variance in each group of the new generated meta examples. We found that the largest variance is found in the meta examples generated based on the aggregation-based approach. For this reason, the aggregation-based approach yields the best mean relative improvement as seen before.

6.3.3. Encoded Representation of Meta Features

Similar to the encoding process presented in Chapter 5, the meta features are encoded in this group of experiments. In the hidden layers, the input values are encoded in such a way that they can be decoded again in the output layer. When a hidden layer has fewer neurons

Table 6.5.: General comparison of the effect of different meta examples generation approaches in terms of mean relative accuracy in the case of original representation of meta features.

Training Set Size	Performance Improvement in (%)			
	Aggregation + Original	Aggregation + Weather + Original	Model + Original	Weather + Original
321	8.75	5.03	6.11	4.8
428	12.32	11.03	12.11	8.46
642	14.13	10.12	13.38	11.67
856	15.60	12.88	**14.67**	11.96
1070	14.41	**14.56**	13.02	12.91
1284	**16.77**	14.00	13.14	11.59
1498	15.16	14.02	11.74	**15.37**

than the input layer, the information in the input layers will have to be compressed leading to a more powerful representation form of features.

In this group of experiments, we investigate the effect of meta examples on the predictive performance of meta learners when using an encoded representation of meta examples. Encoding a meta example includes applying an autoencoder only on the meta features in this meta example. The goal of using autoencoders is to increase the amount of information included in the features, to remove the correlation between them and reduce the dimensionality of the meta learning classification problem. No feature selection procedure is applied in this group of experiments. As a result, we have 67 meta features in our meta examples to be encoded. The autoencoder is implemented using the deeplearning functions from h2o library in R with the following parameter settings: One hidden layer of size 8 is used to compress the information. The number of neurons in the last layer is equal to them in the first layer that corresponds to the number of input meta features we want to encode, namely 67 meta features. The activation function is the hyperbolic tangent (tanh).

The encoded meta-feature vectors are then the outputs of the neurons in the hidden layer of the autoencoder. The resulting encoded meta feature vectors are of length 8 compared to the original ones, which have length 67. The same meta learning use case used in the first group of experiments is used here with the goal of finding the best power generation forecasting model for energy time series datasets presented in Section 6.2.1. We repeat our experiments 100 times and the mean results are presented in this section. Equation (6.1) presented in the first group of experiments is used to calculate the mean relative improvement in the accuracy of the meta learning classification model.

In Figure 6.18, the mean relative performance improvement depending on the aggregation-based approach is shown. As seen in this figure, the predictive performance in assigning the best forecasting model improves dramatically as the original training set is extended with additional meta examples. A performance improvement of 23.56% is achieved by extending the original training set to double its size. Extending the original training set beyond this training set size resulted in results ranging from 22.25% to 27.07%. The best performance improvement of 27.07% was achieved using a training set size of 856 which is

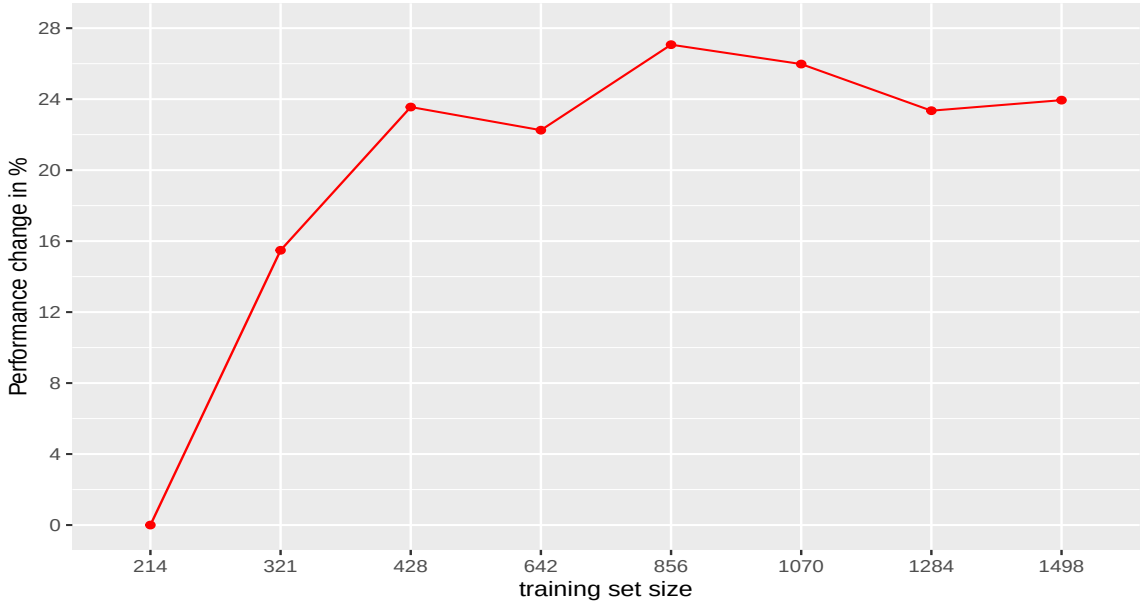


Figure 6.18.: The mean relative accuracy improvement of the meta learner when extending original with encoded aggregation-based datasets.

better than the improvement obtained using the original representation of meta features as seen before.

In Figure 6.19, the mean relative improvement achieved by extending the original training set with the new weather-based meta examples is presented. As seen in this figure, the performance improvement increases for larger training set sizes, with the exception of the training set size 856 which is still better than the performance of the original meta examples, namely 214 meta examples. However, the increases become very small for large training set sizes. The performance improvements for the training set sizes 1070, 1284 and 1498 are 19.17%, 19.43% and 19.80% respectively. The best performance increase of 19.80% is achieved at the maximum training set size of 1498. This is similar to the first experiment where the best result was achieved with the largest training set size of 1498 as well (see Figure 6.15).

The mean relative improvement in the predictive performance of the ANN meta learner related to the model-based approach and a combination of aggregation as well as weather-based approaches are presented in Figure 6.20 and 6.21, respectively. For the model-based approach, the mean performance improvement increases steadily up to 22.16% for a training set size of 642. Increasing the training set size beyond this point resulted in improvement values ranging from 19.73% to 21.57%. Regarding the combined approach, the improvement increases steadily to 20.62% at 642 meta examples as depicted in Figure 6.21. However, the best and second best performances of 23.03% and 21.86% for this approach are achieved using the two largest training set sizes.

Table 6.6 presents a general comparison of the relative performance improvements of extending original meta examples with encoded ones. A good mean relative performance improvement is achieved for all of our approaches compared to the original case that

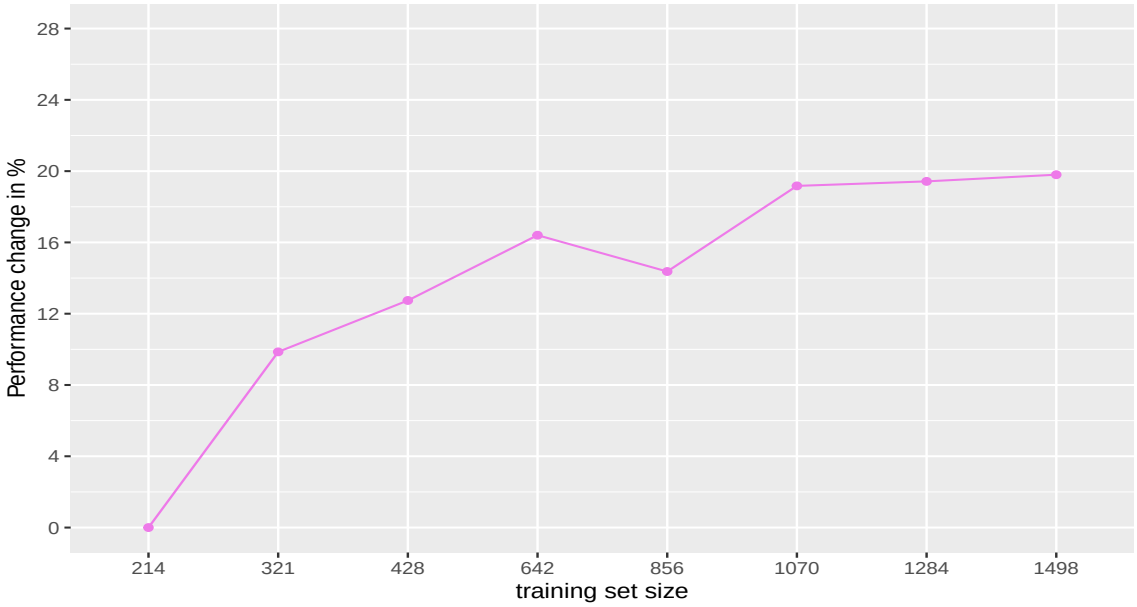


Figure 6.19.: The mean relative accuracy improvement of the meta learner when extending original with encoded weather-based datasets.

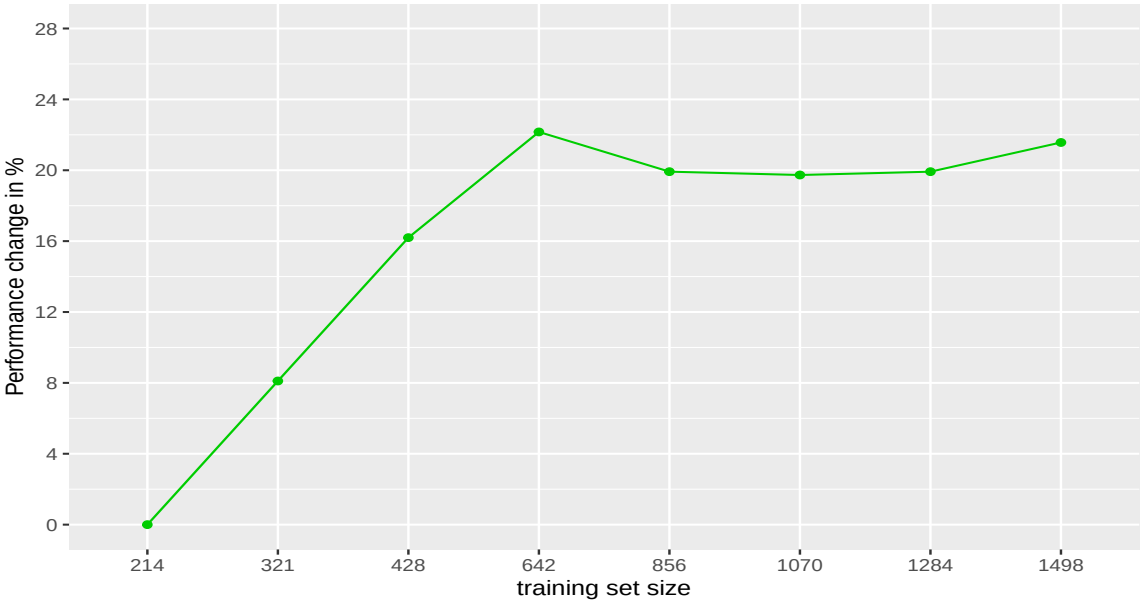


Figure 6.20.: The mean relative accuracy improvement of the meta learner when extending original with encoded model-based datasets.

contains only 214 meta examples. Similar to the first experiment, the additional meta examples in the training set seem to increase the variance in meta features and improve the generalization capabilities of the meta learning classification model.

Interestingly, there is a limit after which no further improvement is obtained. Those limits are different depending on the proposed approach as seen in Table 6.6. E.g., the best results for the weather-based approach are achieved at the largest training set size in both

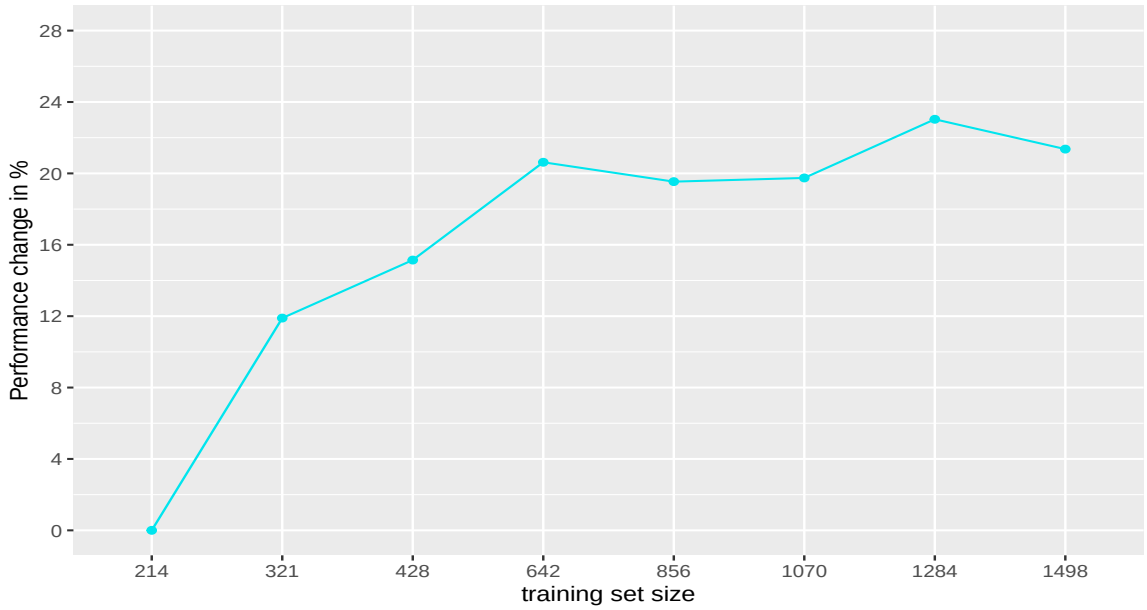


Figure 6.21.: The mean relative accuracy improvement of the meta learner when extending original with combination of encoded aggregation- and weather-based datasets.

experiments. In contrast, for the model-based approach the best results are reached at training set sizes of 642 and 856 in the first group of experiments and at training set sizes of 642 and 1498 in the current group of experiments. The best improvement of 27.07% is achieved for the aggregation-based approach at extending the original dataset to include 856 meta examples. This can be explained by the high variance of the features in the extended meta examples on which the meta learner is trained. For the other methods the best achieved results are only 19.80%, 22.16% and 23.03% for the weather-based, model-based and combination-based approaches respectively.

The usage of autoencoder for encoding the meta features generally introduces better results than the one obtained based on the original representation of meta features. The best achieved overall performance improvement is 27.07%. It is achieved by the aggregation-based meta examples with an increase of 10.30% compared to the result of 16.77% in the first group of experiments.

As seen in Table 6.6, the model-based approach existing in the literature is outperformed by the aggregation-based one for every training set size. In contrast to the aggregation-based method, using the weather-based method resulted in the worst performance improvement out of all methods for 6 out of 7 training set sizes. This is similar to the first experiment, where the weather-based approach produced the worst results out of all approaches for 5 out of 7 extended training set sizes as presented in Table 6.5. Despite that, extending the original meta examples with meta examples resulting from the weather-based approach still improves the performance of the meta learner in comparison the case that only original meta examples are used.

Extending the original training set with a combination of meta examples from both aggregation- and weather-based approaches resulted in a mean performance improvement

Table 6.6.: General comparison of the effect of different encoded meta examples generation approaches.

Training Set Size	Performance Improvement in (%)			
	Aggregation + Original	Aggregation + Weather + Original	Model + Original	Weather + Original
321	**15.48**	11.89	8.11	9.86
428	**23.56**	15.14	16.20	12.74
642	**22.25**	20.62	22.16	16.41
858	**27.07**	19.54	19.92	14.37
1070	**25.98**	19.74	19.73	19.17
1284	**23.35**	23.03	19.92	19.43
1498	**23.94**	21.86	21.57	19.80

that lies between the results of the stand-alone approaches. This can be explained by looking at the variance of the features. Based on the fact that the variance of features plays an essential role in the accuracy of the classification model, we analysed the variance of features in each of the new time series datasets generated from the aggregation-based, weather-based, model-based and a combination of aggregation as well as weather-based approaches. We found that the large variance is existing in the datasets generated by the aggregation-based approach. The lowest variance is found in the datasets generated from the model-based approach.

Moreover, combining the meta examples resulting from both aggregation-based and weather-based introduced a variance which is lower than the variance of the aggregation-based and higher than weather-based datasets. Therefore and for every training set size, the aggregation-based approach performs better and the weather-based approach performs worse than a combination of them. This confirms the results from the first experiment, that combining the meta examples of both approaches didn't improve performance, but rather that the weather-based meta examples dragged down the excellent relative performance improvement achieved by using only aggregation-based meta examples for extending the original one.

6.4. Summary

Meta learning is a machine learning method, which can quickly perform automatic algorithm selection. A so-called meta learner tries to learn the relationship between algorithm performance and problem characteristics using a data-driven approach. Because meta learning is a data-driven approach, the good classification performance of the meta learner depends on the availability of training examples. In this chapter, we introduced new concepts to generate new meta examples. We started in Section 6.1 by presenting the problem we aim to solve. In Section 6.2, we introduced our solution for generating new meta examples.

An extensive evaluation study for measuring the effect of the new generated meta examples on the predictive accuracy of the meta learning classification model is provided in Section 6.3. We analyzed the effects taking two points of view. While the original representation of meta features is used in the first one, an encoded representation of meta features using autoencoder is utilized in the second one. The evaluation results show that the newly generated meta examples are able to enhance the predictive performance of the ANN meta learner up to 16.77% and 27.07% in the case of using the original and encoded representation forms of meta features respectively.

7. Automated Time Series Model Selection in Big Data Environments

With the introduction of more and more renewable generation and volatile load an, efficient management of energy at grid level requires more and more accurate load and generation forecasting. Despite intensive research projects in this field, energy load and generation forecasting still represent a challenging task especially for non-expert users where finding the optimal candidate of a forecasting model can be a very time consuming and complex task.

The well-known no-free-lunch theorem states that no single forecasting model can provide the optimal solution for each forecasting problem [173]. Therefore, depending on the forecasting situation and a given time series to forecast, an appropriate model needs to be found for the given situation. The straightforward method to use a trial-and-error process, in which a lot of learning algorithms with different hyperparameters are tried out till a model with acceptable predictive performance is found, is a tedious and boring process. But even in this process, expert knowledge is highly required to understand, recognize and build the most suitable forecasting model. Moreover, this task is more difficult in the context of Big Data where a large amount of energy time series datasets need to be processed and afterwards forecasted. In order to minimize these difficulties, machine learning algorithms in a so-called meta learning approach can be used to automatically select the best forecasting model for a particular energy time series dataset.

In this chapter, a new microservice-based solution for instrumenting the concept of meta learning presented in Chapters 5 and 6 is introduced. The main advantage of the developed framework is the ability of solving the problem of energy time series model selection in Big Data environments and supporting non-expert users, category 2B, in performing energy time series forecasting for Big Data without the need of having deep knowledge in the field of machine learning and Big Data.

Firstly, Section 7.1 describes the problem statement in more detail. In this section, the problems to be solved, the challenges to be tackled, the group of non-expert users to

Parts of this chapter are reproduced from:

- S. Shahoud, H. Khalloof, M. Winter, C. Duepmeier, and V. Hagenmeyer (2020). "A Meta Learning Approach forAutomating Model Selection in Big Data Environments using Microservice and Container Virtualization Technologies". In: Proceedings of the 12th International Conference on Management of Digital EcoSystems, pp. 84–91. doi: 10.1145/3415958.3433072

be assisted, goals to be achieved, the scientific question to be answered and the major advantages of using microservices techniques for approaching the problem solution are described. Secondly, the conceptual microservice-based architecture for the proposed meta learning solution as an extension of the architecture introduced in Chapter 4 is presented in details in Section 7.2.1. An in-depth experimental study is performed in Section 7.3 to evaluate the performance of the meta learning framework in two points of view. In the first one, the predictive accuracy of the meta learning classification model in assigning the best forecasting model to the input energy time series datasets is evaluated. In the second one, the framework overhead and execution time required for performing meta learning tasks is investigated and discussed. Before starting the discussion of results obtained from the evaluation, the setup of experimental environment and therefore the deployment of the proposed meta learning framework in Big Data Environments is described. The research contributions presented in this chapter were the main topics of our paper in [145].

7.1. Problem Statement

As mentioned before in Chapter 4, a microservice-based framework is developed for supporting non-expert users category 2A presented in Table 1.1 to perform ML tasks in Big Data environments. This framework hides the low level configurations from the user and interfacing to the ML software on the cluster in a way that will allow plugging in different ML runtime environments, e.g. Apache Spark.

While users of category 2A have good know-how in data analytic, another category of non-expert users is existing, when performing ML tasks, namely category 2B, as seen in Table 1.1. Such users are inexperienced in the field of ML and statistics. Moreover, it is difficult or even impossible for them to build a required time series forecasting model, due to the lack of required experience in the field of ML. They only have the energy time series datasets and need to have some accurate forecasting results.

Choosing an adequate ML model for energy time series forecasting is not an easy task for those users. The reason for that lies in the fact that a great number of ML algorithms are existing and further can be applied to solve the same problem, but it's difficult to find the right one. Such algorithms originate from different fields, for example, statistics, time series, neural networks, to name a few. Tuning the hyperparameters of such algorithms to correctly build an accurate forecasting model requires deep expert knowledge in the field of ML. Moreover, this task will become more complex when dealing with a large amount of energy time series datasets. To address this issue, the instrumenting of meta learning is proposed in this thesis to efficiently support non-expert users in selecting the best forecasting model without the need for building and testing a large number of models manually.

The main concept of meta learning presented in Chapter 5 can be instrumented by conceptualizing and implementing a microservice-based meta learning framework running in the Big Data environments which can be used for combining the different building

blocks of our solution into one easy-to-use integrated software platform. The proposed meta learning framework therefore takes advantage of modern runtime techniques such as container and microservices technologies. Microservices represent one of the state-of-the-art software architecture approach, where each service is designed to be a separate deployable software component running in its own execution environment making it horizontally scalable as mentioned in Chapter 2. In contrast to a monolithic architecture where the application is designated as one single unit, the microservice architecture splits up the entire dedicated functionality into a collection of smaller units communicating with each other via lightweight APIs, e.g. REST-APIs. This ensures better scalability, a very modular design and clearly visible explicit inter-dependencies between the different components solely defined by their service API. Thus, new functionality can be easily added to the system by introducing new services. Since each service has its own runtime environment and can be scaled independently from each other, container technologies are used to encapsulate each one in a standardized container image for runtime automation purposes.

In this chapter, we answer the research question RQ4 "How should non-expert users with neither ML knowledge, nor programming skills be supported in performing energy time series forecasting in Big Data environments?" by extending the solution presented in Chapter 4 to include three additional microservices for meta learning, namely Data Preprocessing Service (D.P.-Service), Meta Knowledge Extraction Service (M.K.E.-Service) and Meta Learning Service (M.L.-Service). More details about the general microservice-based architecture are provided in Section 7.2.

The developed framework is designed to be generic and flexible as much as possible for providing automatic model selection to a wide range of application areas. In the present work, energy time series model selection is considered as a meta learning use case as already introduced before. To this aim, a variety of meta features groups are extracted, namely simple, statistical, Information Theoretic (IT), Time Series (TS) and Descriptive Statistics Time-based Meta Features (DSTMF) meta features [143]. Both Random Forecast (RF) and Artificial Neural Network (ANN) algorithms are involved in the proposed meta learning framework as meta learners. To ensure better applicability of the classification rules by the meta learner, an efficient feature selection procedure is also integrated into our solution. The current framework utilizes Apache Spark as a runtime environment for ML on a Big Data cluster and spark.ml as a ML library. The storage layer of the framework utilizes the Hadoop Distributed File System (HDFS) and PostgreSQL database for storing the required input and the resulting output data as will be seen in the following sections.

7.2. Proposed Solution

In order to build a scalable and highly flexible meta learning framework for recommending the most appropriate forecasting model, the framework architecture presented in Chapter 4 is extended. In this section, we present our conceptual meta learning microservice-based architecture proposed to answer the aforementioned research question RQ4. To this end,

we clarify in detail the layers, the microservices and the communication between them to achieve the main goal of our framework.

7.2.1. Conceptual Meta Learning Microservice-based Architecture

Figure 7.1 describes the conceptual microservice-based architecture of our meta learning solution in Big Data environments. The major extension here compared to the architecture presented in Chapter 4 are the automated model selection UI besides the additional services required to realize the main concept of meta learning.

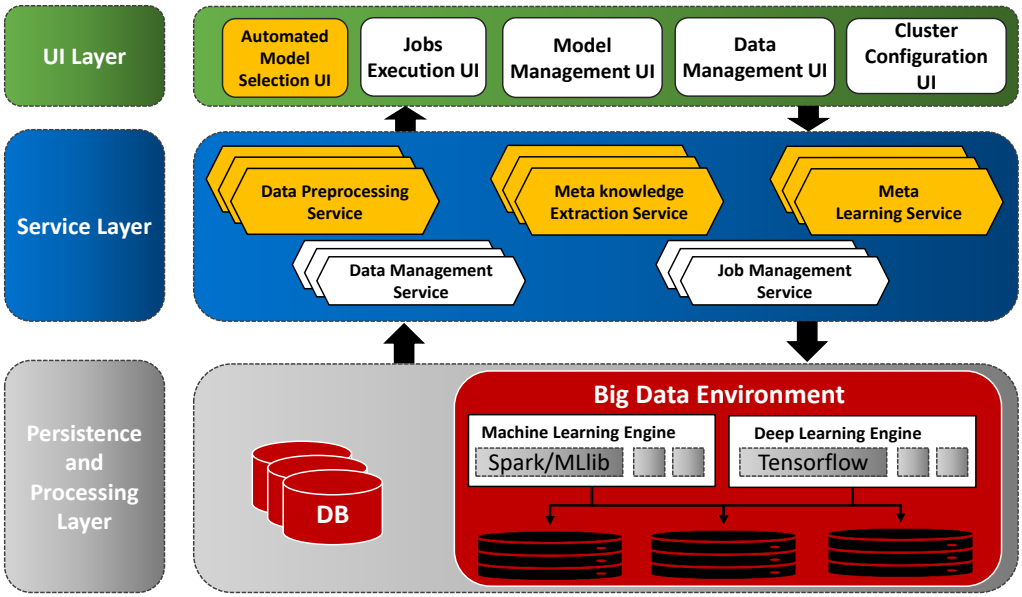


Figure 7.1.: Conceptual framework architecture for meta learning, adapted from Chapter 4.

In the architecture presented in Chapter 4, two microservices are developed to perform ML jobs in Big Data environments. While one service focused on data and model management, the other service is responsible for running and monitoring ML jobs. As mentioned before, the main functionalities of the microservice-based framework presented in Chapter 4 are extended by adding three additional microservices to the service layer, namely data preprocessing, meta knowledge extraction and meta learning services to support non-expert users in selecting an adequate ML model for a given ML task as seen in Figure 7.1. The services provide RESTful APIs which are used by the applications in the UI layers to interact with the runtime environment to perform the main tasks.

Figure 7.2 illustrated the mapping between the meta learning concept and the proposed microservice-based framework. In this mapping, the different steps for building a meta learning classification model are mapped into their corresponding microservices. In the following sections, the layers including the microservices are described in more detail.

Algorithm 1 Meta Learning System

Phase 1: prepare energy time series datasets and build meta learning classification model

Input:

$X = \{x_1, x_2, \ldots, x_n\}$: Problem space (energy time series datasets)
$A = \{a_1, a_2, \ldots, a_m\}$: Algorithm space
$Agg = \{Daily, Weekly, Monthly, \ldots\}$: Aggregation levels of X
PP: Set of functions to preprocess X
MF: Set of functions to extract meta features categories
$DSTMF$: Set of functions to extract DSTMF meta features
FF: Set of functions to calculate forecasting features

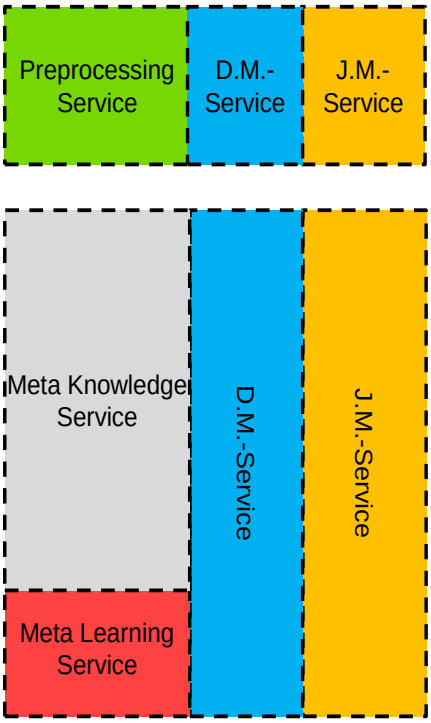
Output:

Meta learning classification model

preprocessing and features extraction of problem space X
for i = 1 to n in X **do**
    Apply PP to preprocess x_i
for i = 1 to n in X **do**
    Calculate forecasting features FF for x_i

Extract Meta knowledge
for i = 1 to n in X **do**
    Split x_i into training set $x_i\_training$ and test set $x_i\_testing$
    for j = 1 to n in Agg **do**
        Aggregate $x_i\_training$ based on Agg
        Calculate meta features $DSTMF$ on the aggregated datasets
    Calculate meta features MF on training set $x_i\_training$
    for k = 1 to m in A **do**
        Train model a_k on $x_i\_training$
        Test model a_k on $x_i\_testing$ and save forecasting error
    Select model with best performance measure as a label for x_i
Build Knowledge Matrix (KM)

Build and evaluate meta learning classifier on KM
Split KM into $KM\_training$ and $KM\_testing$
train and test meta learning classification model on $KM\_training$ and $KM\_testing$



Phase 2: model selection for a new energy time series dataset

Input:

Meta learning classification model
x_{new}: Energy time series dataset to classify
FF and $DSTMF$ functions

Output:

The adequate forecasting model

Apply PP to preprocess time series dataset x_{new}
Aggregate x_{new} based on Agg
Apply $DSTMF$ functions on the aggregated results
Apply MF functions on x_{new}
Predict the best model for x_{new} based on meta features
Update KM with x_{new}

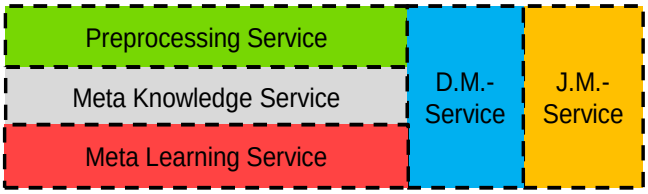


Figure 7.2.: Mapping the different steps of the meta learning approach to the corresponding microservices.

7.2.1.1. User Interface (UI) Layer

This layer handles the user interaction and provides the necessary functionality and information to aid non-expert users in accomplishing their ML tasks in Big Data environments effectively. It consists of separate web applications which interact with the service layer via RESTful APIs to provide the dedicated functionalities. In the present meta learning framework, the UI presented in Chapter 4 is extended to include the automated model selection UI to support non-expert users in performing efficiently selecting the most appropriate energy time series forecasting model. In the following, the sub-parts of the UI are explained in more detail.

Data Management UI: As mentioned in Chapter 4.

Model Management UI: As mentioned in Chapter 4.

Job Execution UI: As mentioned in Chapter 4.

Cluster Configuration UI: As mentioned in Chapter 4.

Automated Model Selection UI: This UI is new and responsible for the tasks of meta learning. It allows the user to upload or select a new dataset, for which the most appropriate ML model needs to be automatically recommended, e.g. in our setting a "time series forecast". After that, a new meta learning task is established. In this task, a meta learning classification model based on a wide range of datasets stored in the persistence and storage layer of the proposed framework is built and used to assign the adequate, e.g. forecasting model, for the new uploaded energy time series dataset.

7.2.1.2. Service Layer

As mentioned before, the main advantage of using microservices, apart from their modularity, is that each component of the framework can be developed separately using, e.g. different technologies and programming languages. In this layer and in the context of meta learning, the main domain functionalities have been divided into five microservices on the service layer, namely the Job Management Service (J.M.-Service), the Data Management Service (D.M.-Service), Data Preprocessing Service (D.P.-Service), Meta Knowledge Extraction Service (M.K.E.-Service) and Meta Learning Service (M.L.-Service) as shown in Figure 7.1. Because both J.M.-Service and D.M.-Service were already explained in Chapter 4, we will only present the other services in more detail in the following sections.

D.P.-Service: As its name implies, this service is responsible for preprocessing and preparing energy time series datasets for the tasks of forecasting and meta learning. The preprocessing steps involved in this service include outlier detection or the handling of gaps in the input energy time series datasets. Forecasting features which are required to perform an accurate forecasting are extracted by this service. This step is followed by an efficient feature selection procedure in which the most important uncorrelated features are selected. Moreover, the aggregation task in DSTMF [143] is considered as a preprocessing step and performed by this service. Synthetic Minority Over-sampling

Technique (SMOTE) and Principal Component Analysis (PCA) techniques as major steps to enhance the predictive performance of the meta learning classification model are applied by this service. Generally, the D.P.-Service communicates with D.M.-Service to get the time series datasets that need to be preprocessed. Then, it creates a preprocessing job and sends it to the J.M.-Service which in turn accesses the persistence and processing layer to execute the job on the Big Data cluster. The main functionalities of D.P.-Service REST-APIs are summarized by the URL patterns presented in Table 7.1.

Table 7.1.: List of the URL patterns of the D.P.-Service.

URL Pattern	Description
/extractFeatures/dataset_id	A post request on this URL is used to Start feature extraction (forecasting features) for dataset and adds features to dataset file
/aggregate/dataset_id	A post request on this URL is used to create new aggregated dataset
/aggregate/dataset_id/all	A post request on this URL is used to create new datasets with all available aggregation levels
/process/dataset_id	A post request on this URL is used to perform preprocessing such as outlier detection or the handling of gap
/selectFeatures/dataset_id	A post request on this URL is used to Start feature selection procedure on the features extracted for this dataset
/pca/dataset_id	A post request on this URL is used to perform principal component analysis
/smote/dataset_id	A post request on this URL is used apply synthetic minority over sampling technique

M.K.E.-Service: As mentioned before, the main goal of meta learning is to support non-expert users in finding the most adequate forecasting model. To this end, meta learning tries to find the mapping between the meta features that describe the energy time series datasets and the algorithm space, from which the best model is recommended. To achieve that, the Knowledge Matrix (KM), as defined in Chapter 5, needs to be built to be later used by the M.L.-Service to perform the time series automated model selection. The M.K.E.-Service is responsible for preparing information for the building of the Knowledge Matrix.

In the KM, each row corresponds to one energy time series dataset containing meta features of it as predictors and the best model e.g. forecasting model as a label. Five different types of meta features are taken into account as already described before, namely Simple, Statistical, Information-theoretic, Time Series and DSTMF meta features. A comprehensive description about them was already is presented in Chapter 5. The main functionalities of M.K.E.-Service REST-APIs are summarized by the URL patterns presented in Table 7.2. Building and maintaining the KM is performed as follows:

1. The M.K.E.-Service communicates with the D.M.-Service to get the required energy time series datasets, from which the meta knowledge will be extracted.
2. The M.K.E.-Service triggers both D.M.-Service and J.M.-Service for trying all possible models on the cluster to determine the best and most adequate one to be set as a label for each time series dataset.
3. The M.K.E.-Service triggers both D.M.-Service and J.M.-Service for extracting the required meta features for each time series dataset.

Table 7.2.: List of the URL patterns of the M.K.E.-Service.

URL Pattern	Description
/metaFeatures	A GET request on this URL is used to retrieve all available meta features
/metaFeaturese/dataset_id	A GET request on this URL is used to retrieve meta features for a specific dataset
/metaFeaturese/dataset_id	A POST request on this URL is used to extract meta features for a specific dataset
/testingPerformance	A GET request on this URL is used to retrieve all available testing performance
/testingPerformance/dataset_id/algo_id	A GET request on this URL is used to retrieve testing performance for a specific dataset and algorithm
/testingPerformance/dataset_id/algo_id	A POST request on this URL is used to execute training for specific dataset with a specific algorithm
/testingPerformance/dataset_id/all	A GET request on this URL is used to retrieve testing performance for all available algorithms for a specific dataset
/testingPerformance/dataset_id/all	A POST request on this URL is used to execute training for specific dataset with all available algorithms in parallel

M.L.-Service: After extracting the meta knowledge by the M.K.E.-Service, the M.L.-Service gather this information to build KM. After that, the M.L.-Service trains and tests an algorithm, referred to as a meta learner, responsible for learning the mapping between meta features and the best models. As meta learners, RF and ANN are used. Based on this mapping learned during the training phases, the resulting meta learning classification model is used later to recommend the best forecasting model for a new energy time series dataset.

To achieve that, the M.L.-Service first sets up the meta learning job, in which a meta learning model is trained and tested. Then, it sends this job to the J.M.-Service which in turns accesses the persistence and processing layer to perform the job on the cluster. The main functionalities of M.L.-Service REST-APIs are described by the following URL patterns described in Table 7.3:

7.2.1.3. Persistence and Processing Layer

This layer is responsible for storing all data and processing tasks. It communicates with the service layer to perform ML tasks in Big Data environments hiding the details of the runtime environments from the implementation of the services. A detailed explanation of this layer was already given in Chapter 4.

7.3. Evaluation

To evaluate the efficiency of the microservice-based meta learning framework, a set of experiments were designed to explore several aspects related to the accuracy of the meta

Table 7.3.: List of the URL patterns of the M.L.-Service.

URL Pattern	Description
/project	A GET request on this URL is used to retrieve all meta learning projects
/project	A POST request on this URL is used to create a new meta learning project
/project/project_id	A GET request on this URL is used to retrieve a specific meta learning project
/project/project_id/dataset_id	A POST request on this URL is used to add a specific dataset to the project
/project/project_id/datasets	A GET request on this URL is used to retrieve all datasets in a specific project
/knowledge/project_id/metaFeatures	A GET request on this URL is used to retrieve all meta features for a specific project
/knowledge/project_id/labels	A GET request on this URL is used to retrieve all class labels for a specific project
/knowledge/project_id/update	A POST request on this URL is used to gather knowledge matrix and upload it to a distributed file system
/knowledge/project_id/metaFeatureCorr	A POST request on this URL is used to calculate meta features correlations
/knowledge/project_id/featureImportance	A POST request on this URL is used to calculate meta features importance scores
/learner/project_id/train	A POST request on this URL is used to train meta learner on knowledge matrix
/learner/project_id/evaluate	A POST request on this URL is used to evaluate meta learner
/learner/project_id/rfe	A POST request on this URL is used to execute recursive feature elimination
/learner/project_id/classify/dataset_id	A POST request on this URL uses a trained meta learner to predict a model for a specific dataset

learner, the execution time and the overhead of the framework. The use case used for performing these experiments is the same one presented in Section 5.2.2, in which a meta learning scenario for selecting the best short-term load forecasting model is presented. In this use case, seven groups of meta features, namely Simple, Statistical, Information-theoretic, Time Series, DSTMF, All without DSTMF and All were instrumented. Random forest and artificial neural networks, as meta learners, are used to discover the mapping between the extracted meta features and the best forecasting model from a group of 4 models, namely decision tree, random forest, gradient boosted tree and linear regression. For better utilization and exploitation of the available abilities of the underlying Big Data cluster, the custom configurations shown in Table 4.3 are used. 200 time series datasets corresponding to 200 buildings are selected from Energo+ in these experiments.

In contrast to the evaluation presented in Section 5.3, in which the accuracy of meta features in characterizing energy time series datasets is measured, the focus here is on the enhancement achieved in the predictive performance of the meta learner. The enhancements are analyzed from three points of view as shown in Figure 7.4. In the first one, the curse of dimensionality that negatively affects the predictive performance

of the meta learning classification model is solved by applying Principal Component Analysis (PCA). In the second one, the high correlation between meta features is removed by applying an efficient feature selection procedure. In the third one, the imbalancing in the resulting knowledge matrix is fixed by applying Synthetic Minority Over-sampling Technique (SMOTE) to increase the number of examples that correspond to minority class.

Moreover, the computational complexity of the different groups of meta features extracted in this work is discussed, highlighting DSTMF as a new convincing form of meta features for characterizing energy time series datasets. The evaluation is started by presenting the deployment of the microservice-based meta learning framework on cluster for setting up the evaluation environment. After that, the obtained results are introduced and discussed.

7.3.1. Deployment of Microservice-based Meta Learning Architecture in Big Data Environments

In this section, a brief overview of the deployment of our proposed microservice-based meta learning framework in Big Data Environments is introduced. For a long time, the term virtualization referred to as a hypervisor-based virtualization. A hypervisor is a software which allows an abstraction from the underlying hardware. It can be used to emulate the hardware as a "virtual machine". This virtual machine runs isolated from the host system and other virtual machines managed by the hypervisor software. Inside the virtual machine, an operating system and software can be installed. In contrast to hypervisor-based virtualization, container-based virtualization does not emulate the necessary hardware but uses operating system features to isolate processes and create so called containers as isolated runtime environments for applications.. All containers of a system use the same operating system kernel which significantly lowers the overhead.

Docker is the most well-known container runtime environment, where each docker software installation consists of multiple components:

- **Docker Daemon:** It can be seen as a background daemon process running on the host operating system which is accessible from other computers by exposing a REST API, and otherwise. It is also responsible for creating and managing the running containers on the host.
- **Docker Container:** It can be seen as a standardized isolated runtime environment on a host which runs applications.
- **Docker Image:** It can be seen as a template for setting up an internal file system within a container to run a dedicated set of applications. Therefore, images can be used to store and deploy applications with all their third party dependencies needed for running the applications (e.g. frameworks, configuration files, needed content, etc.). The file system representing the image contain all software which is needed to run one or more applications within the container.

- **Registries:** It can be seen as a repository for Docker images. Images can be pushed to or pulled from registries by the docker daemon to execute them in a container on a host. Registries can either be public or private, and make images available for easy deployment.

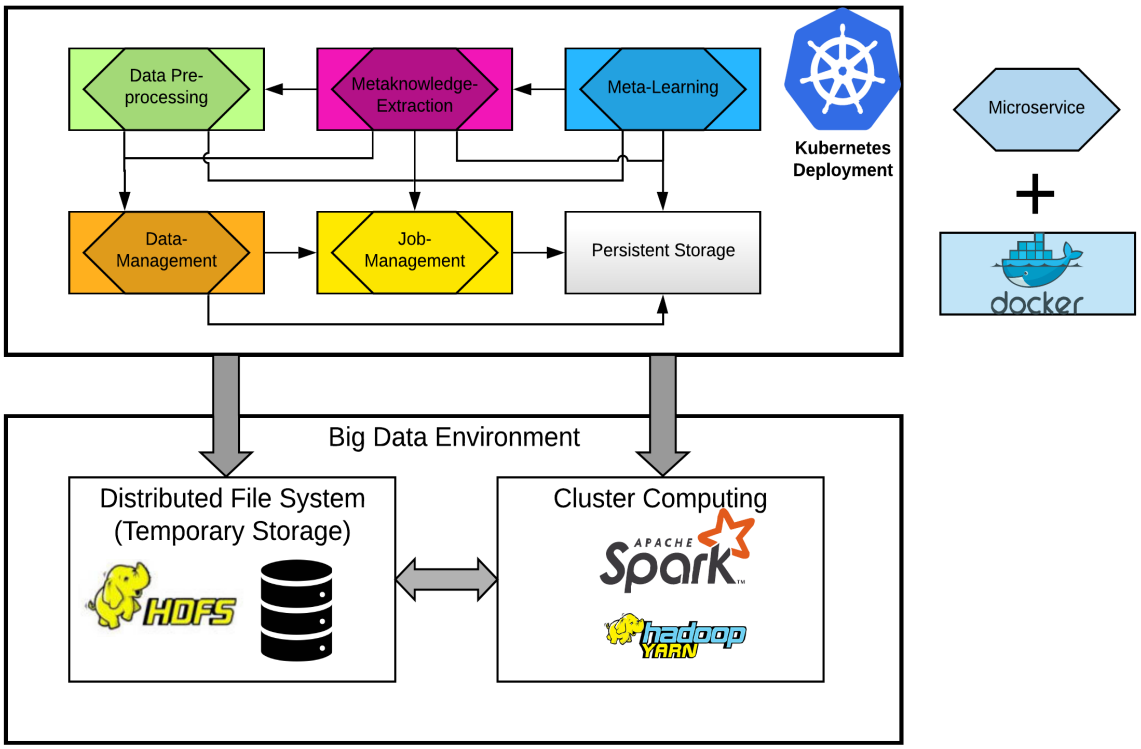


Figure 7.3.: Deployment of Microservices.

Figure 7.3 illustrates the deployment of our meta learning framework on cluster. As seen in this figure, docker was utilized to build container environments for the microservice presented in Section 6.2.2. For each microservice, we create a docker image that can be accessed by a private docker registry. This procedure allows us to exploit the advantages our microservice architecture entails, namely easy re-deployment and scalability. The persistent database was installed as a PostgreSQL server that also resides in a docker container.

To automatically deploy docker containers on different nodes in a cloud environment, a container orchestration software, called Kubernetes, is used. Kubernetes allows managing multiple machines (nodes) as a cluster. Each cluster contains at minimum one master node and one worker node. The master node is responsible for global decisions about the cluster and monitors the state of worker nodes. It also provides an interface for access by clients.

Each worker node contains multiple components:

- **Kubelet:** It is an agent that makes sure all pods are healthy. A pod is the simplest kubernetes object and represents a set of running containers. The Kubelet takes pod specifications and tries to fill them.
- **Kube-proxy:** It manages network rules on nodes.
- **Container Runtime:** It is the software that is responsible for running the containers, for example Docker.

Using kubernetes as container orchestration software, it is possible to deploy multiple containers and specify the network environment, persistent storage or replications and can pull docker images directly from a docker registry. Therefore, it is well-suited for the deployment of multiple microservices which have to communicate with each other.

7.3.2. Results and Discussion

In this section, the predictive performance of the proposed microservice-based meta learning framework in recommending the best forecasting model for energy time series datasets is evaluated. As mentioned before, the same meta learning use case presented in Chapter 5 is considered in this group of experiments. The goal is to enhance the predictive performance of the meta learner by applying state-of-the-art techniques in feature engineering. Firstly, the obtained results in terms of the accuracy of the meta learning classification model are presented and discussed. After that, the framework overhead and execution time are introduced.

7.3.2.1. Accuracy of Meta Learning Classification Model

Figure 7.4 illustrates the overall methodology applied in the evaluation for achieving the best predictive performance of the meta learner using the aforementioned groups of meta features. As seen in this figure, an efficient feature selection process including the removal of highly correlated features followed by Recursive Feature Elimination (RFE) is performed to achieve better classification results.

To address the problem of imbalanced classes, Synthetic Minority Over-sampling Technique (SMOTE) [34] is applied to create a more balanced dataset. The curse of dimensionality that is considered as one of the major problems affecting the accuracy of the classification models is mainly addressed by applying a Principal Component Analysis (PCA), whereby PCA is one of the state-of-the-art unsupervised linear transformation technique that learns the relationships between predictors to find a list of principal components [129]. To this end, the number of features is reduced leading to more accurate classification results as will be seen in this section. All of these techniques are performed by the aforementioned M.K.E.-Service and the D.P.-Service.

In this section, only the results of applying Principal Component Analysis (PCA) and Synthetic Minority Over-sampling Technique (SMOTE) are introduced, as the other results

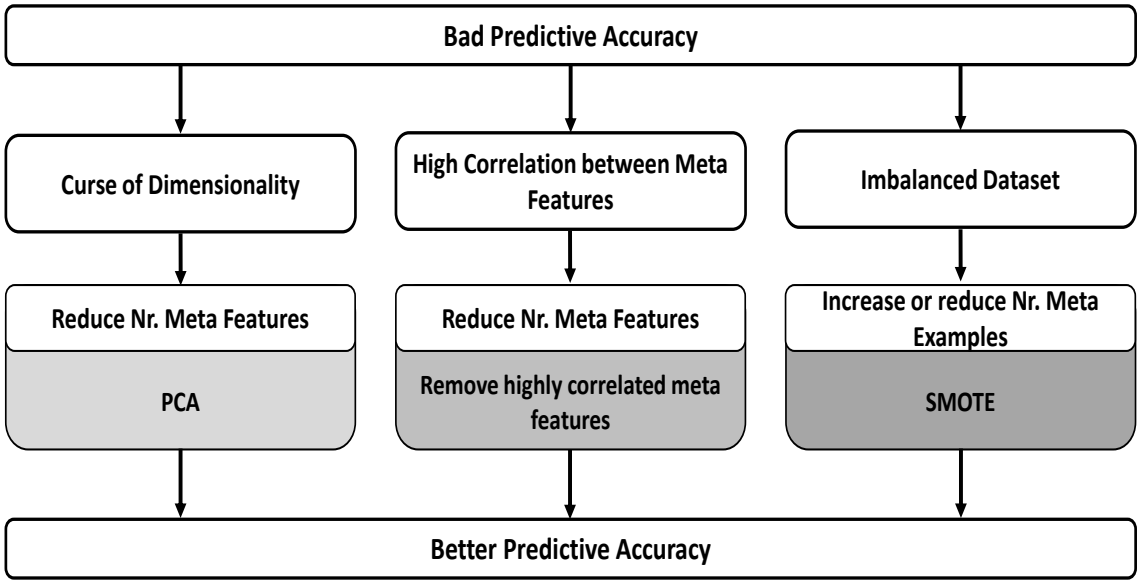


Figure 7.4.: Methodology applied to improve the predictive performance of meta learning classification model.

related to removing the highly correlated meta features were already presented and discussed in Section 5.3 for our use case.

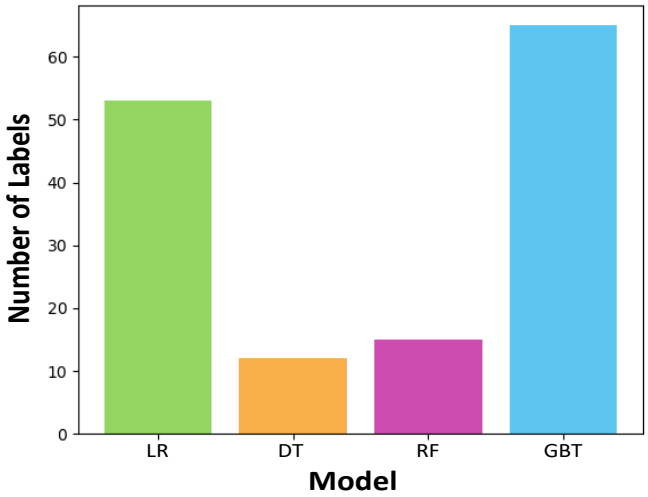


Figure 7.5.: Model Distribution before applying SMOTE.

After extracting meta features and building the required short-term forecasting models for each input time series dataset to set up the knowledge matrix, the distribution of models is shown in Figure 7.5. As seen in this figure, the Gradient Boosted Tree (GBT) models introduced the best forecasting predictive performance for the greatest number of time series datasets, closely followed by the Linear Regression models. Random Forest and Decision Tree Models rank not quite as good as the other models. Such distribution

of labels finally leads to an imbalanced knowledge matrix which will badly affect the predictive performance of our meta learning classification model.

To address this issue, oversampling techniques are applied. SMOTE is one of the state-of-the-art techniques existing in the literature to perform oversampling, in which the number of minority classes is increased to achieve the balance in the overall class distribution. The main idea is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced class distribution. The class distribution of DT, RF, DT and GBT after applying SMOTE is shown in Figure 7.6. As seen in this figure, the number of minority classes in the original unbalanced dataset is increased to become 62 and 48 for DT and RF respectively, where GBT still represents the majority class with 65 values against 53 values for the class of LR.

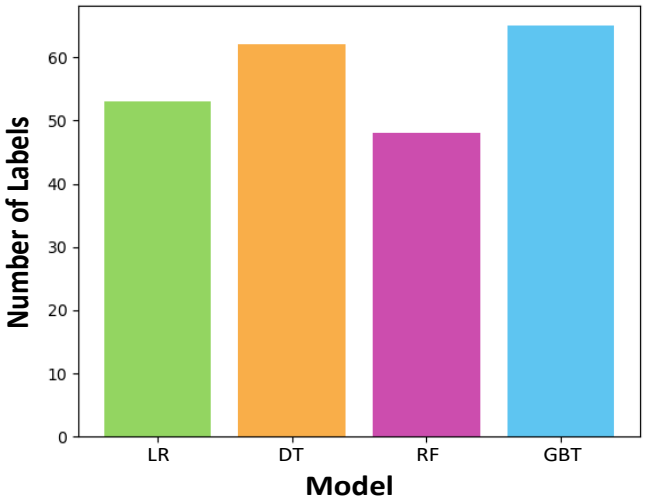


Figure 7.6.: Model Distribution after applying SMOTE.

The predictive performance of both RF and ANN as meta learners are presented in Tables 7.4 and 7.5 respectively. As seen in these tables, both meta learners achieved a classification accuracy of 80% in the third testing scenario in which the testing dataset is not covered by the training one.

Table 7.4.: The predictive performance of random forest meta learner after applying SMOTE technique.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Partly	10	40	20	90	90	80	70
Complete	10	40	20	80	90	80	80
Not Covered	10	40	20	70	80	70	70

Table 7.5.: The predictive performance of neural network meta learner after applying SMOTE technique.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Partly	20	40	20	70	90	70	70
Complete	10	40	20	90	90	90	90
Not Covered	10	30	10	60	80	60	70

Table 7.6.: The predictive performance of random forest meta learner after applying PCA technique.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Partly	10	40	20	80	80	80	60
Complete	10	40	20	90	90	90	90
Not Covered	10	40	20	60	70	60	80

Another technique proposed in literature to solve the curse of dimensionality and multi-collinearity between features is PCA. The goal of these experiments is to select the most efficient meta features and reduce the total number by producing new efficient components of meta features. Generally, PCA is a statistical procedure that uses an orthogonal transformation to convert a set of possibly correlated features into a set of features of linearly uncorrelated variables called principal components. The disadvantage in such approaches lies in the fact that new components are introduced to represent the input meta features hiding as well as encapsulating the original meta features in a new group of features. Therefore, we will not be able to recognize the final meta features used in our meta learning classification model.

Before applying PCA, it is highly recommended to normalize the dataset. The reason behind this is that performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance. Concerning the number of principal components, we decided to produce the same number of meta features resulted from the feature selection process presented in Table 5.4. That means, 2, 7, 2, 20, 22, 30 and 52 principal components for the 7 meta-feature groups, namely Simple, Statistical, Information-theoretic, TS, DSTMF, All without" and All, respectively as seen in Tables 7.6 and 7.7.

The resulting principal components represent the original meta features and are used to train and test the performance of random forest and neural network meta learners in the aforementioned three different testing scenarios. Tables 7.6 and 7.7 clearly show the obtained results. As seen in these tables, in the third testing scenario, a classification accuracy of 80% for both random forest and neural network is achieved when using all available meta features. In the last meta feature category, the positive effect of DSTMF can

Table 7.7.: The predictive performance of neural network meta learner after applying PCA technique.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Partly	10	40	20	80	80	80	70
Complete	10	40	20	90	90	90	90
Not Covered	10	50	20	70	70	60	80

be clearly seen achieving an improvement of 20% compared to the category "All without DSTMF" for both meta learners.

Typically, the third testing scenario is the most difficult and reliable one compared to the other testing scenarios considered in these experiments. Driven by that, only the predictive accuracy of RF and ANN meta learners in the case of a testing dataset that is not covered by the training one is summarized.

Table 7.8.: General Comparison in the case of using random forest meta learner.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Without Processing	10	40	30	60	50	40	50
Feature Selection	10	40	20	80	80	60	80
SMOTE	10	40	20	70	80	70	70
PCA	10	40	20	60	70	60	80

Table 7.9.: General Comparison in the case of using neural network meta learner.

	Predictive Accuracy						
	Simple	**Statistical**	**Information-theoretic**	**TS**	**DSTMF**	**All Without**	**All**
Without Processing	10	40	30	50	60	40	60
Feature Selection	10	40	20	90	90	70	90
SMOTE	10	30	10	60	80	60	70
PCA	10	50	20	70	70	60	80

It is clearly seen in Tables 7.8 and 7.9 that handling Simple, Statistical and Information-theoretic separately don't lead to an acceptable classification accuracy for either of the both meta learners. Even after using the feature selection, SMOTE and PCA, both meta learners introduce a bad predictive performance in recommending the adequate short-term forecasting model when using these groups of meta features as seen in Tables 7.8 and 7.9. While RF and ANN achieved an accuracy of 70% for TS and DSTMF meta features, the best predictive performance in terms of accuracy was achieved when using all meta features to characterize energy time series datasets. Comparing the previous tables, it is highly recommended to use neural networks as meta learners to find the association between meta features and the best forecasting model.

7.3.2.2. Framework Execution Time and Overhead

So far, the accuracy of the meta learning classification model is evaluated on different categories of meta features. In this section, the efficiency of our proposed microservice-based meta learning framework in recommending the best forecasting model is investigated. To achieve that, the total time required for different tasks is measured. The experiments are repeated five times and the mean values of the results are computed.

Case 1: Meta Feature Extraction: The goal of this task is to extract the meta features required to describe energy time series datasets. As already discussed, these meta features are used as predictors in the meta learning classification model. This task is performed by the M.K.E.-Service but there is also some inter-service communication needed in order to get information from the D.M.-Service and submit the job via the J.M.-Service.

The total time needed for this task is calculated according to Equation (7.1) and it is subdivided into an overhead and extraction time as shown in Figures 7.7 and 7.8 respectively.

$$T_{total} = T_{ms} + T_{co} + T_{spark} + T_{ex} + T_{preparation} \tag{7.1}$$

Where:
T_{ms} is the internal processing time of the microservices including database access, T_{co} is the time the microservices need for their internal communication, T_{spark} is the time needed to launch the spark application added to Spark overhead, T_{ex} is the total time required within the cluster to perform the task, $T_{preparation}$ is the time required to load the dataset from HDFS for processing, where each dataset is loaded only one time and all required operations are performed on it.

As seen in Figure 7.7, the microservice processing time and inter-service communication overhead is vanishingly small and we have a relative constant spark application launch overhead. The time needed to load and prepare the dataset increases with greater dataset sizes, which is also to be expected.

A look at the meta feature extraction time in Figure 7.8 reveals a great increase in the processing time, as the size of the dataset increases. To investigate further how the extraction time is distributed between the calculations of the different groups of meta features, the extraction time is measured for the calculation of each group separately as depicted in Figure 7.9.

Interesting is that DSTMF introduces a comparative predictive performance to TS as seen in Section 7.3.2.1 with the advantage that DSTMF exhibits a much lower extraction time than TS.

Case 2: Forecasting Feature Extraction: Calendar features such as hour, day, month and weekend are extracted. To this aim, the D.P.-Service prepares a Spark application with the respective needed information and communicates with both D.M.-Service and

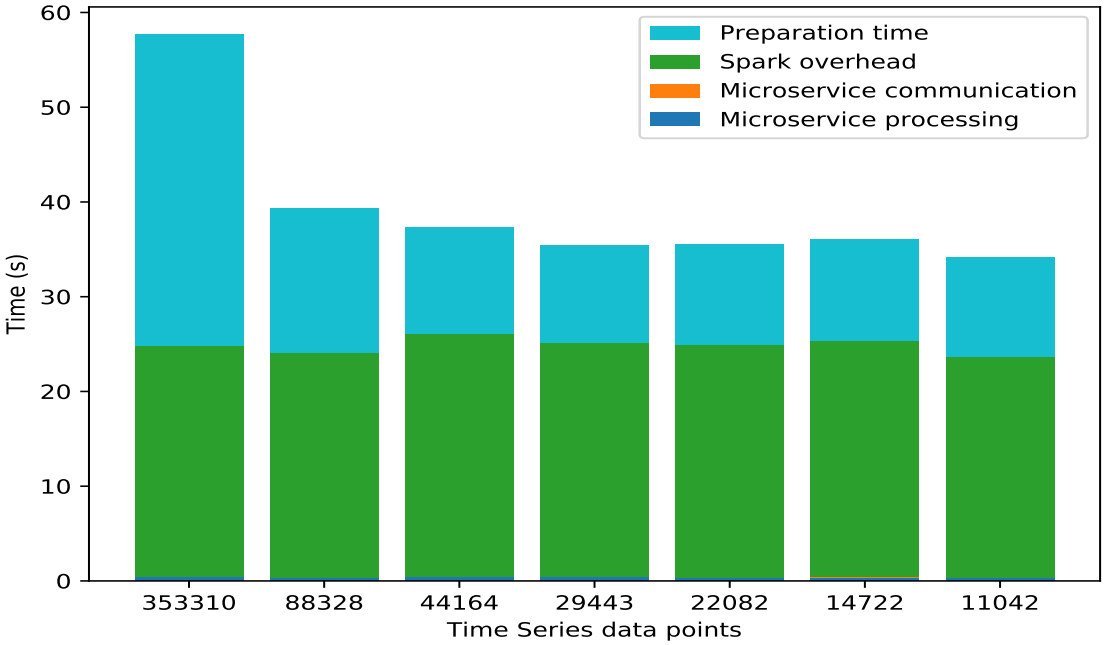


Figure 7.7.: Meta feature extraction: overhead.

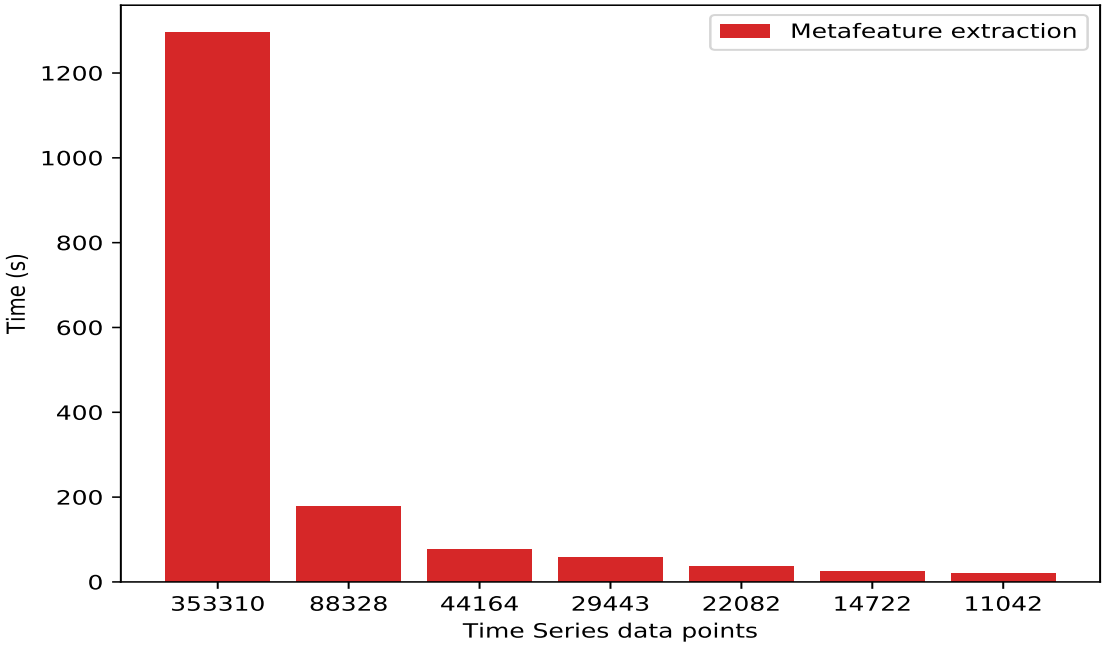


Figure 7.8.: Meta feature extraction: execution time.

J.M.-Service to launch the Spark application and send it to the cluster. The total time is calculated according to Equation (7.2).

$$T_{total} = T_{ms} + T_{co} + T_{spark} + T_{ex} \tag{7.2}$$

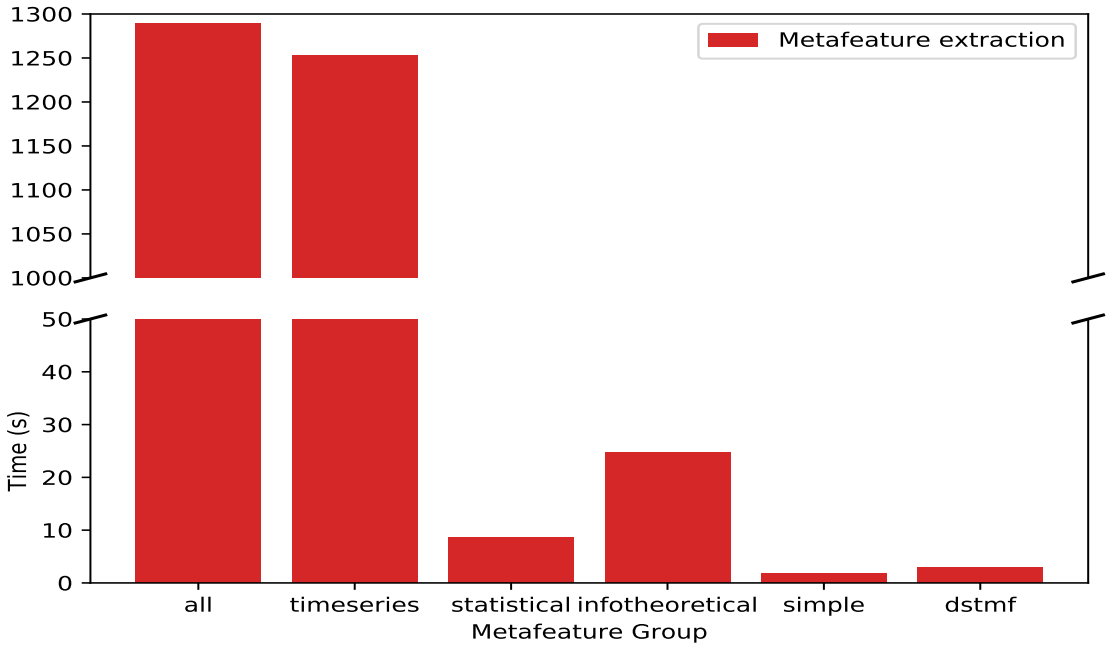


Figure 7.9.: Meta feature extraction: extraction time by meta feature groups.

To evaluate the effect of dataset size, the measurements are performed for different aggregation levels of the same dataset. The original time series has an aggregation level of 15 minutes and 353310 data points which is a load time series of over 10 years. The other compared datasets are created by the following aggregation levels: 1H (1 Hour), 2H, 3H, 4H, 6H and 8H. It can be observed that compared to the execution time and the Spark overhead, the microservice processing and communication time are quite small and do not really impact the T_{total} much as seen in Figure 7.10.

Case 3: Building the Meta Learning Classification Model: After constructing the KM by the M.K.E.-Service, the M.L.-Service trains RF and ANN classifiers as meta learners to build the meta learning classification model. This model is responsible for assigning the best forecasting model for a new energy time series dataset without the need for trying a lot of forecasting algorithms to discover the best one. The total time is calculated according to the previous Equation (7.2).

The building of the meta learning model can be done either in evaluation mode or production mode. While the M.L.-Service performs a 10-fold cross validation in the evaluation mode, it builds the meta learning model by splitting the KM into training and testing without cross validation in the production mode. The mean T_{total} in both evaluation and production mode is presented in Figure 7.11 for the RF meta learner, where the large bars correspond to the evaluation mode reflecting the additional time required for training the meta learning model compared to a smaller time needed in case of production mode. This experiment is carried out using different sizes of KM as seen in Figure 7.11, where no difference in the training time of RF is noticed in the production mode. This is due to the power of Apache Spark, for which a KM of 1000 examples is considered as a small dataset

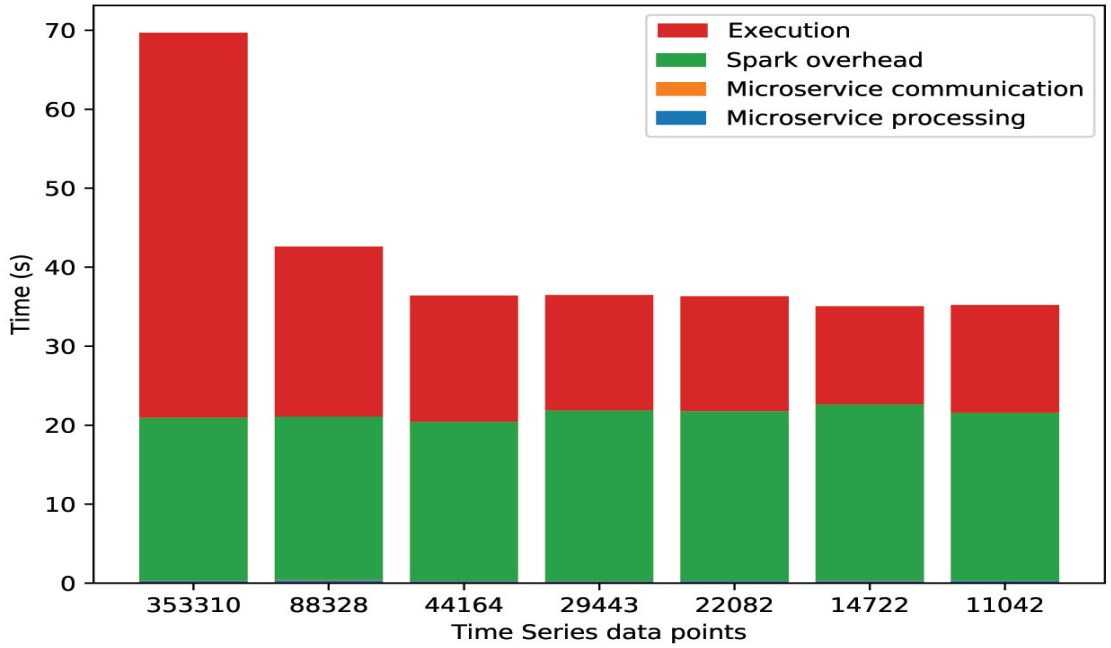


Figure 7.10.: Forecasting feature extraction time.

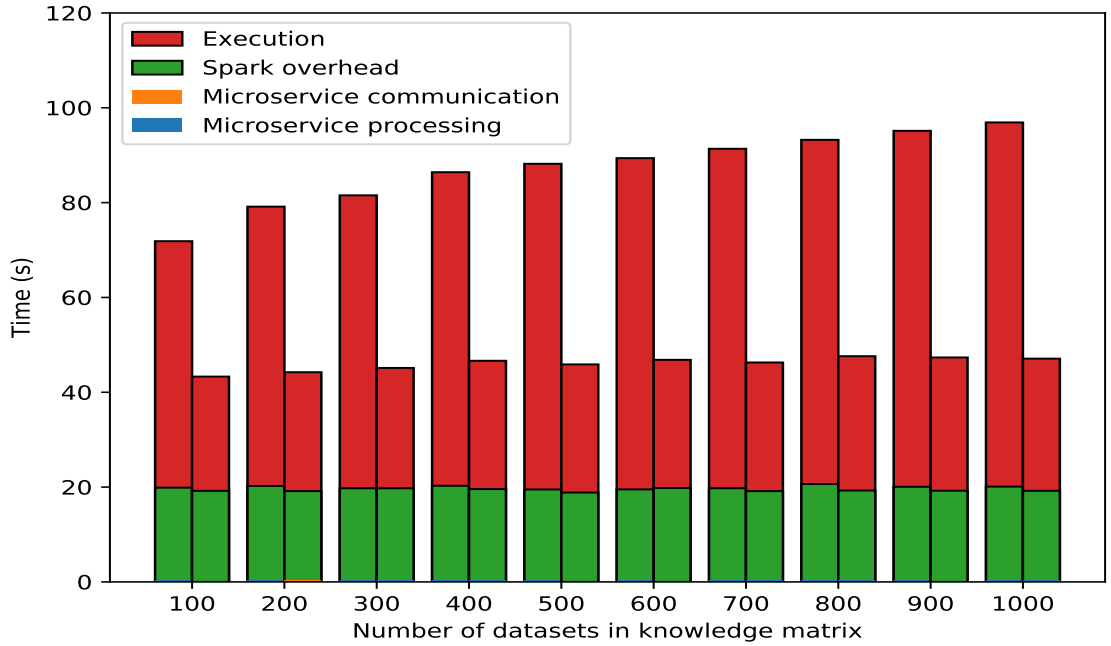


Figure 7.11.: The mean T_{total} in both evaluation and production modes required for building RF meta learning classification model.

and does not affect the execution time. For the evaluation mode, the higher execution time can be attributed to the cross validation process which trains not only one but multiple models to get an average classification accuracy over all models.

7.4. Summary

In this chapter, a microservice-based meta learning solution for automating energy time series model selection in Big Data environments is presented. It is designated as a modular, scalable and generic tool based on container and microservice technologies. In Section 7.1, problem statements are introduced focusing on problems to be solved, techniques to be used and non-expert users to be supported. In Section 7.2.1, the conceptual meta learning microservice-based architecture is introduced highlighting the extension that is performed upon the architecture presented in Chapter 4 to realize the concept of meta learning.

An extensive evaluation study was carried out to measure the predictive accuracy of meta learning classification model and framework execution time as well as the framework overhead as presented in Section 7.3. The obtained results are discussed in Section 7.3.2.1 including the effect of different groups of meta features on the predictive performance of meta learners. In this context, the predictive performance of the meta learning classification model is improved by applying an efficient feature engineering procedure. To further improve the predictive performance, SMOTE and PCA techniques are applied. The evaluation results showed that an accuracy of 80% is achieved for both RF and ANN meta learners.

In terms of execution time and overhead, the proposed framework introduced an acceptable execution time with small overhead as seen in Section 7.3.2.2. It should be mentioned that the new type of meta features introduced, namely DSTMF, have, besides good characterization behaviour of energy time series datasets, also a very small execution time and overhead compared to the other categories of meta features.

8. Summary and Outlook

In this chapter, we present a short summary of the solutions, concepts and methodologies proposed in this work for supporting non-expert users in performing ML tasks and arrive at some main conclusions. Afterwards, we will give some suggestions regarding potential future work.

8.1. Summary

Applying Machine Learning (ML) manually to a given problem setting is a tedious and time-consuming process which brings many challenges with it, especially in the context of Big Data. With the growing interest in ML the last few years, particularly people without extensive ML expertise have a high demand for frameworks assisting people in applying the right ML algorithm to their problem setting. This is especially true in the field of smart energy system applications where more and more ML algorithms are used e.g. for time series data forecasting or state estimation.

The aim of this thesis was to evaluate how meta learning technology can be efficiently applied to the problem space of data analytics for smart energy systems to assist energy system experts which are not data analytics experts in applying the right ML algorithms to their data analytics problems. The work was performed in context of the research work for conceptualizing and implementing a digital research platform for German energy researchers within the Helmholtz research program ESD. Thus, there was a high demand that results of this work could be seamlessly integrated as an expandable solution into this bigger digital research platform. Therefore, main building blocks of such an integrable solution were conceptualized, and then implemented as parts of an evaluation platform: (1) e.g. an easy-to-use modular extendable microservice-architecture based ML framework for instrumenting and evaluating ML algorithms performed by using Open Source Big Data ML software stacks running on computing clusters with a modular web based user interface (2a) a generic meta learner framework extending this environment (2b) a generic concept of how meta features could be represented by meta feature sets which are unfortunately dependent on the given data analytics problem type, e.g. forecasting load or generation time series, in a generic fashion. Therefore, a generic solution was introduced where meta feature sets could be "parameters" of the problem to be solved. Especially for evaluation, the dedicated set of meta feature DSTMF was instrumented as new art of meta features for the ML task of time series forecasting (3) how the concept of a knowledge matrix together with the concept of a meta feature set can be used to train the generic meta learner which

then finds the most adequate ML algorithm for a given data analytics job, e.g. time series forecasting (4) how data have to be pre-processed so that they can be used by the meta learning building blocks (5) how the learning environment can be widen and enhanced by automatically generating additional data sets for training (6) How a software environment with these software components can be setup with user interfaces that allows the user to perform complex data analytics tasks on computing clusters without expert knowledge in either data analytics or cluster computing environments.

To evaluate the building blocks by performing experiments and gathering evaluation data, concrete use cases were instrumented on the evaluation platform with a focus on performing time series analysis and forecasting on load and generation times series data sets. Using the instrumented use cases, the evaluation platform was then used to perform many data analytics experiments for gathering evaluation data capturing key characteristics of the building blocks of the meta learning platform. These evaluation data were then analyzed to verify main statements of the proposed research questions introduced in the introduction of this thesis. In following, the stated research questions and the main evaluation results are summarized.

The first research question stated in the introduction "How should non-expert users with ML knowledge, but without programming skills be supported in performing ML tasks on computing clusters using Big Data ML frameworks?" is answered in Chapter 4 by proposing a microservice- based framework built on top of big data software stack to ensure the ability of performing ML tasks for large amounts of data on computing clusters. Our conceptual solution facilitates the well-known trial-and-error approach in which a combination of different algorithms with their hyperparameters are tried by an ML user (non-expert users, category 2A introduced in the introduction) until an adequate model is found. This is done by supporting this trial-and-error approach by an easy-to-use software tool environment which can be used via a web browser and where several approaches can run in parallel on the cluster computing environment. The proposed ML framework can even cache tuned models, and their results for easier replication or tuning. It connects a modular highly configurable Web UI in the front-end with a grid of microservices in the back-end which in turn communicates with distributed Big Data ML processing environments via their service interfaces. The framework decouples the user from the Big Data runtime environment, and sets up the runtime environment automatically to execute ML jobs based on user input. It thereby hides the complexities of the Big Data runtime environment from the user. The whole environment is highly scalable, modular, expandable and needlessly integrable with other microservice-based environments, such as the digital energy research platform of the ESD program.

The framework was evaluated using different sizes of energy time series datasets and four state-of-the-art regression algorithms for time series regression forecasting implemented in spark.ml, namely linear regression, decision trees, gradient-boosted trees, and random forests as examples of forecasting algorithms. The framework is designed in such a way that further algorithms can be easily added to the framework as needed. As seen in Chapter 4, the major advantage of the proposed framework lies in the microservice-based architecture that ensures better scalability and maintainability. Moreover, the framework

has the ability to manage, store and retrieve ML models to be used later for another task without the need to build a new model. The results of the evaluation showed only a small framework overhead and an obvious benefit of utilizing a cluster environment for larger data sets. Caching of input data is another advantage of our framework that has a positive effect of speeding up the execution of the ML tasks while introducing only a low overhead. Also, the thresholds as well as the conditions to which the input data must adhere, so that performing the ML tasks on clusters is feasible, were introduced and discussed.

To support even non-expert users category 2B, a meta learning solution was conceptualized and integrated into the ML environment (see steps 2-5 further above). This solution was evaluated in context of several further research questions stated in the introduction. The second research question "How to efficiently characterize energy time series datasets to enhance the performance of automated model selection?" introduced in Chapter 5 proposed the Descriptive Statistics Time based Meta Features (DSTMF), a new form of meta feature for better capturing the deep characteristics of energy time series datasets regarding their forecasting. DSTMF is based on the idea of aggregating energy time series datasets into different aggregation levels as "forecasting horizons with different time resolution". After that and for each level, the five number summary in addition to the mean are extracted and formulated together as a group of meta features to capture the energy time series "time sequential" behavior.

As seen in Chapter 5, a short-term load forecasting scenario was used for evaluation. In our evaluation study, DSTMF outperformed the other state- of-the-art meta features introduced in the literature to characterize energy time series datasets, namely simple, statistical, information-theoretic and time series meta features. We used random forest and artificial neural networks as meta learners to learn the mapping between meta features and the forecasting models. As a result of our evaluation, the artificial neural network solution introduced better predictive performance in recommending the best forecasting model than random forest. The predictive performance was analyzed in two different cases. In the first one, the original representation form of meta features is used in which only an efficient feature selection procedure on meta features is applied. In the second one, unsupervised deep learning using an autoencoder is applied to encode the meta feature into another more compact representation form. One result is that the encoded representation form outperformed the original one. Additionally, the meta learner was able to assign the best forecasting model even when it is trained on a few number of training examples.

The third research question "How to enhance the performance of automated model selection in the context of energy by creating appropriate learning datasets?" was investigated in Chapter 6. Two new approaches to generate new energy time series datasets to be used as training meta examples by the meta learner depending on the type of time series dataset (i.e. generation or energy consumption time series) were developed. The first approach for generation datasets makes use of weather time series datasets to split, filter, sort and group the input energy time series datasets into different "snippets" according to their similarity based on the same weather conditions. The snippets can then be used to generate new generation time series by alternating the series of weather conditions in

reasonable ways. The second approach used for energy consumption time series follows the same methodology of DSTMF proposed in Chapter 5.

It aggregates the input energy time series datasets creating new ones with different aggregation levels which is feasible for consumption data because energy is an integral property. We compared the achieved learning performance introduced by the new energy time series datasets with the one achieved by a model-based generation of new datasets as existing in the literature whereby a one-day-ahead power generation forecasting scenario is used as a use case study to evaluate the efficiency of our proposed dataset generation approaches. For such scenarios K-Nearest-Neighbor (KNN), Neural Network (NN), Autoregressive Integrated Moving Average Model (ARMIA) and Exponential Smoothing (ETS) are possible ML forecasting models which were instrumented for the evaluation. The evaluation study was carried out using the original and encoded representation of meta features. Tables 8.1 and 8.2 summarize the final results in terms of enhancing the predictive performance of the meta learner when the training set is scaled up with additional meta examples generated according to our proposed approaches.

Table 8.1.: Relative performance improvement in terms of accuracy achieved by using extended datasets in the case of using original representation of meta features.

Training Set Size	Aggregation + Original	Weather + Original	Aggregation+ Weather+ Original	Model+Original
321	8.88	4.34	5.03	6.11
428	12.32	8.46	11.03	12.11
642	14.13	11.67	10.12	13.38
856	15.60	11.96	12.88	14.67
1070	14.41	12.91	14.56	13.02
1284	16.77	11.59	14.00	13.14
1498	15.16	15.37	14.02	11.74

As seen in this table, extending the original training sets with new meta examples generated by the aggregation and weather-based approaches outperformed the case in which the original is extended by new simulated energy time series datasets. One likely explanation could be that the simulation model based generation approach adheres to a certain mode logic which hinders the creation of too diverse new datasets which makes it more difficult for the meta learner to learn the mappings of all possible forecasting possibilities (even those which are not addressed by the simulation model). The max enhancement is achieved in the case of encoding meta features by autoencoder as seen in Table 8.2 whereby the aggregation-based approach introduces the best case.

The fourth research question "How should non-expert users with neither ML knowledge, nor programming skills be supported in performing energy time series forecasting in Big Data environments?" is answered in Chapter 7. We extended the microservice-based solution presented in Chapter 4 to support the automated selection of the best forecasting

Table 8.2.: Relative performance improvement in terms of accuracy achieved by using extended datasets in the case of using encoded representation of meta features.

Training Set Size	Aggregation + Original	Weather + Original	Aggregation+ Weather+ Original	Model+Original
321	15.48	9.86	11.89	8.11
428	23.56	12.74	15.14	16.20
642	22.25	16.41	20.62	22.16
856	27.07	14.37	19.54	19.92
1070	25.98	19.17	19.74	19.73
1284	23.35	19.43	23.03	19.92
1498	23.94	19.80	21.36	21.57

model. To this end, we combined the new approaches for extracting DSTMF meta features and generating new meta examples into the solution developed in Chapter 4. Additionally, a wizard-like modular web based user interface was created which assists non-expert users, category 2B, in performing ML tasks by automatically selecting the best algorithm using the meta learning environment. A detailed in-depth evaluation study was performed in which the predictive accuracy of the meta learner was investigated. As meta learners, Random Forests and Artificial Neural Networks are used whereby the neural network solution features better performance. We have seen that an efficient feature selection procedure including the resolving of the problem of curse of dimensionality, removing the highly correlated meta features and balancing the training datasets plays an essential role in enhancing the predictive performance of the meta learning classification model. We find that an accuracy of 80% is achieved for both RF and ANN meta learners. We also evaluated the developed framework in terms of execution time and overhead while performing the major tasks in model selection. We have seen that an acceptable execution time with small overhead is introduced. Interesting is that our new form of meta features for solving the ASP problem for forecasting, namely DSTMF, could be calculated in very small execution time and overhead compared to the other categories of meta features. This shows that it is very important to provide for each ML task that should be supported on the platform the right set of meta features for having the best performance.

8.2. Outlook

This thesis introduced a solution to support non-expert users in performing energy power generation and load forecasting in Big Data environments. The proposed solution is an ongoing work for developing an even more interactive and intelligent concept for fully automating, managing, deploying, monitoring, organizing and documenting ML tasks. By answering the research questions presented in Chapter 1, a variety of related research projects are existing in the literature as seen in Chapter 3. However, further

investigation and even more deeper evaluation is still required to precisely capture the possible potentials of our solution. In the following, we highlight some possible future work and extension scenarios concerning each research question.

Research Question 1 [RQ1]:How should non-expert users with ML knowledge, but without programming skills be supported in performing ML tasks on computing clusters using Big Data ML frameworks?

In this context, an in-depth user feedback study with a larger number of users, in particular, non-expert users, should be performed to gather feedback for tuning to make the web-based user interface as easy and comfortable as possible for this user group. Toward better management of ML tasks, user authentication and authorization issues have to be taken into account too. Driven by the large advantage of deep learning in a lot of application fields, such as object recognition and anomaly detection, a Big Data deep learning framework as plugable engine should be integrated in the persistence and storage layer to support performing deep learning tasks.

Research Question 2 [RQ2]:How to efficiently characterize energy time series datasets to enhance the performance of automated model selection?

The outlook here includes opening our scope to use our methodological approaches for other forecasting models and scenarios. Not only short-term load forecasting for buildings but also mid- as well as long-term forecasting horizons for different use cases in energy should be handled and included in a benchmark evaluation study. To ensure better applicability of meta feature encoding, the evaluation needs to be carried out and extended to cover PCA and the other arts of autoencoders proposed for the purpose of learning representation. In general, the research on finding adequate meta feature sets should also be extended to other ML tasks beside forecasting, e.g. outlier or anomaly detection.

Research Question 3 [RQ3]: How to enhance the performance of automated model selection in the context of energy by creating appropriate learning datasets?

Further evaluation including other energy use cases is needed to underpin the general usability of the weather and aggregation-based approaches for generating new energy time series datasets. Not only the enhancement of the predictive performance achieved by extending the training set with the new generated meta examples should be measured, but also the properties, similarity and the diversity of the new datasets needs to be investigated to precisely capture their characteristics and differentiate them from the original ones. As the main concept proposed to answer this research question is based on generating new energy time series datasets from the original one, the initial number of the available datasets needs to be precisely defined.

Research Question 4 [RQ4]: How should non-expert users with neither ML knowledge, nor programming skills be supported in performing energy time series forecasting in Big Data environments?

Toward more support of this group of non-expert users, the automation of further steps in the machine learning pipeline, namely preprocessing and hyperparameter tuning needs to be provided. The added deep learning framework should be augmented by already

available deep learning algorithms for performing common ML tasks by non-expert users of category 2B. In general, more ML solutions for common specific ML tasks (e.g. anomaly detection in energy time series data) should be added to the platform as plug & play solutions for non-experts users. Moreover, extending the evaluation study to cover other use cases not only in energy, but also in a wide range of application fields is required to ensure better support of non-expert users from different domains in selecting the best ML model.

Bibliography

[1] O. Adeoye and C. Spataru (2019). "Modelling and forecasting hourly electricity demand in West African countries". In: *Applied Energy*, vol. 242, pp. 311–333. DOI: 10.1016/j.apenergy.2019.03.057 (cit. on pp. 1, 14).

[2] C. Aderaldo, N. Mendonça, C. Pahl, and P. Jamshidi (2017). "Benchmark requirements for microservices architecture research". In: *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, pp. 8–13. DOI: 10.1109/ECASE.2017.4 (cit. on p. 21).

[3] A. Agrawal (2018). "Application of Machine Learning to Computer Graphics". In: *IEEE Computer Graphics and Applications*, vol. 38, no. 04, pp. 93–96. DOI: 10.1109/MCG.2018.042731662 (cit. on p. 1).

[4] M. Ahmed, R. Seraj, and S. Islam (2020). "The k-means algorithm: A comprehensive survey and performance evaluation". In: *Electronics,* vol. 9, no. 8, p. 1295. DOI: 10.3390/electronics9081295 (cit. on p. 15).

[5] E. Akanksha, N. Sharma, and K. Gulati (2021). "Review on reinforcement learning, research evolution and scope of application". In: *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 1416–1423. DOI: 10.1109/ICCMC51019.2021.9418283 (cit. on p. 16).

[6] A. Ali, B. Gabrys, and M. Budka (2018). "Cross-domain meta-learning for time-series forecasting". In: *Procedia Computer Science*, vol. 126, pp. 9–18. DOI: 10.1016/j.procs.2018.07.204 (cit. on p. 70).

[7] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. Aljaaf (2020). "A systematic review on supervised and unsupervised machine learning algorithms for data science". In: *Supervised and unsupervised learning for data science*, pp. 3–21. DOI: 10.1007/978-3-030-22475-2_1 (cit. on p. 14).

[8] E. Almeshaiei and H. Soltan (2011). "A methodology for electric power load forecasting". In: *Alexandria Engineering Journal,* vol. 50, no. 2, pp. 137–144. DOI: 10.1016/j.aej.2011.01.015 (cit. on p. 79).

[9] F. Almonacid, C. Rus, P. Pérez-Higueras, and L. Hontoria (2011). "Calculation of the energy provided by a PV generator. Comparative study: Conventional methods vs. artificial neural networks". In: *Energy,* vol. 36, no. 1, pp. 375–384. DOI: 10.1016/j.energy.2010.10.028 (cit. on p. 79).

[10] E. Alpaydin (2020). *Introduction to Machine Learning* (cit. on p. 13).

[11] F. Alvarez, A. Troncoso, J. Riquelme, and J. Ruiz (2010). "Energy time series forecasting based on pattern sequence similarity". In: *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 8, pp. 1230–1243. DOI: 10.1109/TKDE.2010.227 (cit. on p. 70).

[12] S. Aman, Y. Simmhan, and V. Prasanna (2011). "Improving energy use forecast for campus micro-grids using indirect indicators". In: *2011 IEEE 11th International Conference on Data Mining Workshops*, pp. 389–397. DOI: 10.1109/ICDMW.2011.95 (cit. on pp. 1, 14, 70).

[13] D. Angeline (2013). "Association rule generation for student performance analysis using apriori algorithm". In: *The SIJ Transactions on Computer Science Engineering & its Applications (CSEA)*, vol. 1, no. 1, pp. 12–16. DOI: 10.9756/SIJCSEA/V1I1/01010252 (cit. on p. 15).

[14] J. Antonanzas, N. Osorio, R. Escobar, R. Urraca, F. Martinez-de-Pison, and F. Antonanzas-Torres (2016). "Review of photovoltaic power forecasting". In: *Solar energy*, vol. 136, pp. 78–111. DOI: 10.1016/j.solener.2016.06.069 (cit. on p. 76).

[15] D. Arize and T. Rios (2019). "A comparison study on time series forecasting given smart grid load uncertainties". In: *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 257–262. DOI: 10.1109/BRACIS.2019.00053 (cit. on pp. 116, 117).

[16] J. Armstrong (2001). "Should we redesign forecasting competitions?" In: *International Journal of Forecasting*, vol. 17, pp. 542–545 (cit. on pp. 5, 8, 69).

[17] N. Al-Azzam and I. Shatnawi (2021). "Comparing supervised and semi-supervised machine learning models on diagnosing breast cancer". In: *Annals of Medicine and Surgery*, vol. 62, pp. 53–64. DOI: 10.1016/j.amsu.2020.12.043 (cit. on p. 16).

[18] A. Balalaie, A. Heydarnoori, and P. Jamshidi (2016). "Microservices architecture enables devops: Migration to a cloud-native architecture". In: *IEEE Software*, vol. 33, no. 3, pp. 42–52. DOI: 10.1109/MS.2016.64 (cit. on p. 21).

[19] A. Balalaie, A. Heydarnoori, and P. Jamshidi (2016). "Migrating to cloud-native architectures using microservices: an experience report". In: *Advances in Service-Oriented and Cloud Computing*, vol. 567, pp. 201–215. DOI: 10.1007/978-3-319-33313-7_15 (cit. on p. 4).

[20] P. Baldi (2012). "Autoencoders, unsupervised learning, and deep architectures". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. Vol. 27, pp. 37–49 (cit. on p. 92).

[21] A. Banks and E. Porcello (2017). *Learning React: functional web development with React and Redux* (cit. on p. 44).

[22] M. Belkin, D. Hsu, S. Ma, and S. Mandal (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15849–15854. DOI: 10.1073/pnas.1903070116 (cit. on p. 15).

[23] E. Bisong (2019). "Kubeflow and kubeflow pipelines". In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 671–685. DOI: 10.1007/978-1-4842-4470-8_46 (cit. on pp. 6, 31, 36).

[24] D. Borthakur (2007). "The hadoop distributed file system: Architecture and design". In: *Hadoop Project Website* (cit. on p. 19).

[25] P. Brazdil, J. van Rijn, C. Soares, and J. Vanschoren (2022). *Metalearning: Applications to Automated Machine Learning and Data Mining*. DOI: 10.1007/978-3-030-67024-5 (cit. on pp. 5, 69, 70).

[26] R. Brewer and P. Johnson (2010). "WattDepot: An open source software ecosystem for enterprise-scale energy data collection, storage, analysis, and visualization". In: *2010 First IEEE International Conference on Smart Grid Communications*, pp. 91–95. DOI: 10.1109/SMARTGRID.2010.5622023 (cit. on pp. 3, 6, 28, 36).

[27] A. Candel, V. Parmar, E. LeDell, and A. Arora (2016). "Deep learning with H2O". In: *H2O. ai Inc*, pp. 1–21 (cit. on pp. 3, 6, 31, 36).

[28] G. Casolla, S. Cuomo, V. Di Cola, and F. Piccialli (2019). "Exploring unsupervised learning techniques for the Internet of Things". In: *IEEE Transactions on Industrial Informatics,* vol. 16, no. 4, pp. 2621–2628. DOI: 10.1109/TII.2019.2941142 (cit. on p. 15).

[29] C. Castiello, G. Castellano, and A. Fanelli (2005). "Meta-data: Characterization of input features for meta-learning". In: *International Conference on Modeling Decisions for Artificial Intelligence*, pp. 457–468. DOI: 10.1007/11526018_45 (cit. on pp. 8, 76).

[30] T. Cerny, M. Donahoo, and M. Trnka (2018). "Contextual understanding of microservice architecture: current and future directions". In: *ACM SIGAPP Applied Computing Review,* vol. 17, no. 4, pp. 29–45. DOI: 10.1145/3183628.3183631 (cit. on p. 4).

[31] V. Cerqueira, L. Torgo, and C. Soares (2019). "Machine learning vs statistical methods for time series forecasting: Size matters". In: *arXiv preprint arXiv:1909.13316*. DOI: 10.48550/arXiv.1909.13316 (cit. on p. 114).

[32] T. Chai and R. Draxler (2014). "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature". In: *Geoscientific model development,* vol. 7, no. 3, pp. 1247–1250. DOI: 10.5194/gmd-7-1247-2014 (cit. on pp. 17, 80).

[33] S. Chan, T. Stone, K. Szeto, and K. Chan (2013). "Predictionio: a distributed machine learning server for practical software development". In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2493–2496. DOI: 10.1145/2505515.2508198 (cit. on pp. 3, 6, 29, 36).

[34] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer (2002). "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research*, vol. 16, pp. 321–357. DOI: 10.1613/jair.953 (cit. on p. 142).

[35] C. Chen and C. Zhang (2014). "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data". In: *Information sciences*, vol. 275, pp. 314–347. DOI: 10.1016/j.ins.2014.01.015 (cit. on p. 18).

[36] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob (2015). "Microsoft azure". In: *New York, NY, USA:: Apress*, pp. 3–26. DOI: 10.1007/978-1-4842-1043-7 (cit. on pp. 36, 37).

[37] J. Cruz and D. Wishart (2006). "Applications of machine learning in cancer prediction and prognosis". In: *Cancer informatics*, vol. 2, pp. 59–77. DOI: 10.1177/117693510600200030 (cit. on p. 14).

[38] C. Cui, T. Wu, M. Hu, J. Weir, and X. Li (2016). "Short-term building energy model recommendation system: A meta-learning approach". In: *Applied energy*, vol. 172, pp. 251–263. DOI: 10.1016/j.apenergy.2016.03.112 (cit. on pp. 1, 8, 14, 32, 34, 37).

[39] L. De Lauretis (2019). "From monolithic architecture to microservices architecture". In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 93–96. DOI: 10.1109/ISSREW.2019.00050 (cit. on p. 21).

[40] J. Dean and S. Ghemawat (2008). "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM,* vol. 51, no. 1, pp. 107–113. DOI: 10.1145/1327452.1327492 (cit. on p. 19).

[41] N. Dmitry and S. Manfred (2014). "On micro-services architecture". In: *International Journal of Open Information Technologies,* vol. 2, no. 9, pp. 24–27 (cit. on p. 22).

[42] O. ElTayeby, D. John, P. Patel, and S. Simmerman (2013). "Comparative case study between D3 & Highcharts on Lustre metadata visualization". In: *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pp. 127–128. DOI: 10.1109/LDAV.2013.6675172 (cit. on p. 45).

[43] H. Eskandari, M. Imani, and M. Moghaddam (2021). "Convolutional and recurrent neural network based model for short-term load forecasting". In: *Electric Power Systems Research*, vol. 195, pp. 107–121. DOI: 10.1016/j.epsr.2021.107173 (cit. on pp. 1, 14, 70).

[44] M. Fard, T. Thonet, and E. Gaussier (2020). "Deep k-means: Jointly clustering with k-means and learning representations". In: *Pattern Recognition Letters*, vol. 138, pp. 185–192. DOI: 10.1016/j.patrec.2020.07.028 (cit. on p. 15).

[45] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter (2019). "Auto-sklearn: Efficient and Robust Automated Machine Learning". In: *Automated Machine Learning: Methods, Systems, Challenges*, pp. 113–134. DOI: 10.1007/978-3-030-05318-5_6 (cit. on pp. 5, 10, 31, 33).

[46] R. Fielding (2000). *Architectural styles and the design of network-based software architectures* (cit. on p. 23).

[47] G. Forestier, F. Petitjean, H. Dau, G. Webb, and E. Keogh (2017). "Generating synthetic time series to augment sparse datasets". In: *2017 IEEE international conference on data mining (ICDM)*, pp. 865–870. DOI: 10.1109/ICDM.2017.106 (cit. on pp. 9, 34, 97).

[48] B. Fulcher and N. Jones (2014). "Highly comparative feature-based time-series classification". In: *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3026–3037. DOI: 10.1109/TKDE.2014.2316504 (cit. on p. 76).

[49] K. Gai and M. Qiu (2018). "Optimal resource allocation using reinforcement learning for IoT content-centric services". In: *Applied Soft Computing*, vol. 70, pp. 12–21. DOI: 10.1016/j.asoc.2018.03.056 (cit. on p. 16).

[50] A. Gandomi and M. Haider (2015). "Beyond the hype: Big data concepts, methods, and analytics". In: *International journal of information management*, vol. 35, no. 2, pp. 137–144. DOI: 10.1016/j.ijinfomgt.2014.10.007 (cit. on p. 3).

[51] X. Glorot, A. Bordes, and Y. Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics,* vol. 15, pp. 315–323 (cit. on p. 92).

[52] J. González Ordiano, S. Waczowicz, M. Reischl, R. Mikut, and V. Hagenmeyer (2017). "Photovoltaic power forecasting using simple data-driven models without weather data". In: *Computer Science-Research and Development,* vol. 32, no. 1-2, pp. 237–246. DOI: 10.1007/s00450-016-0316-5 (cit. on pp. 75, 79).

[53] T. Hastie, R. Tibshirani, and J. Friedman (2009). "Random forests". In: *The elements of statistical learning*, pp. 587–604. DOI: 10.1007/978-0-387-84858-7_15 (cit. on p. 56).

[54] T. Hong and S. Fan (2016). "Probabilistic electric load forecasting: A tutorial review". In: *International Journal of Forecasting,* vol. 32, no. 3, pp. 914–938. DOI: 10.1016/j.ijforecast.2015.11.011 (cit. on p. 79).

[55] R. Hyndman (2001). "It's time to move from what to why". In: *International Journal of Forecasting,* vol. 17, no. 1, pp. 567–570 (cit. on pp. 5, 69).

[56] H. Jabbar and R. Khan (2015). "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)". In: *Computer Science, Communication and Instrumentation Devices*, vol. 70, pp. 163–172 (cit. on p. 15).

[57] I. Jakob (1995). "10 usability heuristics for user interface design". In: *Nielsen Norman Group*, vol. 1 (cit. on p. 44).

[58] D. Jalligampala, R. Lalitha, T. Ramakrishnarao, K. Mylavarapu, and K Kavitha (2022). "Efficient Classification of Heart Disease Forecasting by Using Hyperparameter Tuning". In: *Applications of Artificial Intelligence and Machine Learning*, pp. 115–125. DOI: 10.1007/978-981-19-4831-2_10 (cit. on p. 85).

[59] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. Ojea, E. Solowjow, and S. Levine (2019). "Residual reinforcement learning for robot control". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029. DOI: 10.1109/ICRA.2019.8794127 (cit. on p. 16).

[60] A. Johanson, S. Flögel, C. Dullo, and W. Hasselbring (2016). "OceanTEA: exploring ocean-derived climate data using microservices". In: *6th International Workshop on Climate Informatics, NCAR Technical Note NCAR/TN*, pp. 25–28 (cit. on pp. 3, 28, 36).

[61] M. Jordan and T. Mitchell (2015). "Machine learning: Trends, perspectives, and prospects". In: *Science,* vol. 349, no. 6245, pp. 255–260. DOI: 10.1126/science.aaa8415 (cit. on p. 13).

[62] A. Kadhim (2019). "Survey on supervised machine learning techniques for automatic text classification". In: *Artificial Intelligence Review,* vol. 52, no. 1, pp. 273–292. DOI: 10.1007/s10462-018-09677-1 (cit. on p. 14).

[63] A. Kalousis and T. Theoharis (1999). "Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection". In: *Intelligent Data Analysis*, vol. 3, no. 5, pp. 319–337. DOI: 10.3233/IDA-1999-3502 (cit. on p. 31).

[64] Y. Kang, R. Hyndman, and F. Li (2020). "GRATIS: GeneRAting TIme Series with diverse and controllable characteristics". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal,* vol. 13, no. 4, pp. 354–376. DOI: 10.1002/sam.11461 (cit. on pp. 9, 35, 98).

[65] Y. Kang, R. Hyndman, and K. Smith-Miles (2017). "Visualising forecasting algorithm performance using time series instance spaces". In: *International Journal of Forecasting*, vol. 33, no. 2, pp. 345–358. DOI: 10.1016/j.ijforecast.2016.09.004 (cit. on pp. 9, 34, 35, 98).

[66] B. Karlik and A. Olgac (2011). "Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems,* vol. 1, no. 4, pp. 111–122 (cit. on p. 92).

[67] K. Kentaro and T. Yoshimasa (2009). "Identification of the dual action antihypertensive drugs using tfs-based support vector machines". In: *Chem-Bio Informatics Journal*, vol. 9, pp. 41–51. DOI: 10.1273/cbij.9.41 (cit. on p. 14).

[68] A. Khan, A. Laghari, and S. Awan (2021). "Machine learning in computer vision: A review". In: *EAI Endorsed Transactions on Scalable Information Systems*, vol. 8, no. 32. DOI: 10.4108/eai.21-4-2021.169418 (cit. on p. 1).

[69] I. Khan, A. Akber, and Y. Xu (2019). "Sliding window regression based short-term load forecasting of a multi-area power system". In: *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–5. DOI: 10.1109/CCECE.2019.8861915 (cit. on p. 70).

[70] I. Khan, X. Zhang, M. Rehman, and R. Ali (2020). "A literature survey and empirical study of meta-learning for classifier selection". In: *IEEE Access*, vol. 8, pp. 10262–10281. DOI: 10.1109/ACCESS.2020.2964726 (cit. on pp. 5, 31, 97).

[71] M. Khanum, T. Mahboob, W. Imtiaz, H. Ghafoor, and R. Sehar (2015). "A survey on unsupervised machine learning algorithms for automation, classification and maintenance". In: *International Journal of Computer Applications,* vol. 119, no. 13, pp. 34–39. DOI: 10.5120/21131-4058 (cit. on p. 15).

[72] S. Kim and H. Kim (2016). "A new metric of absolute percentage error for intermittent demand forecasts". In: *International Journal of Forecasting,* vol. 32, no. 3, pp. 669–679. DOI: 10.1016/j.ijforecast.2015.12.003 (cit. on p. 17).

[73] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, and S. Corlay (2016). *Jupyter Notebooks-a publishing format for reproducible computational workflows.* Pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87 (cit. on pp. 28, 36).

[74] N. Koluguri, M. Kumar, S. Kim, C. Lord, and S. Narayanan (2020). "Meta-learning for robust child-adult classification from speech". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8094–8098. DOI: 10.1109/ICASSP40776.2020.9053251 (cit. on p. 31).

[75] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown (2019). "Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA". In: *Automated Machine Learning*, pp. 81–95. DOI: 10.1007/978-3-030-05318-5_4 (cit. on pp. 10, 32).

[76] M. Kück, S. Crone, and M. Freitag (2016). "Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data". In: *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 1499–1506. DOI: 10.1109/IJCNN.2016.7727376 (cit. on pp. 9, 31).

[77] C. Kuster, Y. Rezgui, and M. Mourshed (2017). "Electrical load forecasting models: A critical systematic review". In: *Sustainable cities and society*, vol. 35, pp. 257–270. DOI: 10.1016/j.scs.2017.08.009 (cit. on p. 79).

[78] N. Kyurkchiev and S. Markov (2015). "Sigmoid functions: some approximation and modelling aspects". In: *LAP LAMBERT Academic Publishing, Saarbrucken*, vol. 4 (cit. on p. 92).

[79] R. Larsen (1985). "Box-and-whisker plots". In: *Journal of Chemical Education,* vol. 62, no. 4, p. 302 (cit. on p. 69).

[80] D. Le (2020). "Machine learning-based approaches for disease gene prediction". In: *Briefings in functional genomics*, vol. 19, no. 5-6, pp. 350–363. DOI: 10.1093/bfgp/elaa013 (cit. on pp. 14, 20).

[81] C. Lemke, M. Budka, and B. Gabrys (2015). "Metalearning: a survey of trends and technologies". In: *Artificial intelligence review,* vol. 44, no. 1, pp. 117–130. DOI: 10.1007/s10462-013-9406-y (cit. on pp. 5, 69, 97).

[82] C. Lemke and B. Gabrys (2010). "Meta-learning for time series forecasting and forecast combination". In: *Neurocomputing*, vol. 73, 10-12, pp. 2006–2016. DOI: 10.1016/j.neucom.2009.09.020 (cit. on p. 34).

[83] C. Li, J. Wang, L. Wang, L. Hu, and P. Gong (2014). "Comparison of Classification Algorithms and Training Sample Sizes in Urban Land Classification with Landsat Thematic Mapper Imagery". In: *Remote. Sens.*, vol. 6, no. 2, pp. 964–983. DOI: 10.3390/rs6020964 (cit. on p. 97).

[84] Y. Liang, D. Niu, and W. Hong (2019). "Short term load forecasting based on feature extraction and improved general regression neural network model". In: *Energy*, vol. 166, pp. 653–663. DOI: 10.1016/j.energy.2018.10.119 (cit. on p. 70).

[85] S. Lin, I. Yuan-chin, and W. Yang (2009). "Meta-learning for imbalanced data and classification ensemble in binary classification". In: *Neurocomputing*, vol. 73, no. 1-3, pp. 484–494. DOI: 10.1016/j.neucom.2009.06.015 (cit. on p. 31).

[86] X. Lü, T. Lu, C. Kibert, and M. Viljanen (2015). "Modeling and forecasting energy consumption for heterogeneous buildings using a physical–statistical approach". In: *Applied Energy*, vol. 144, pp. 261–275. DOI: 10.1016/j.apenergy.2014.12.019 (cit. on p. 79).

[87] M. Maher and S. Sakr (2019). "Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms". In: *EDBT: 22nd International Conference on Extending Database Technology*. DOI: 10.5441/002/edbt.2019.54 (cit. on pp. 10, 33).

[88] S. Maisto, B. Martino, and S. Nacchia (2019). "From monolith to cloud architecture using semi-automated microservices modernization". In: *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 638–647. DOI: 10.1007/978-3-030-33509-0_60 (cit. on p. 21).

[89] Y. Manichaikul and F. Schweppe (1979). "Physically based industrial electric load". In: *IEEE Transactions on Power Apparatus and Systems,* vol. PAS-98, no. 4, pp. 1439–1445. DOI: 10.1109/TPAS.1979.319346 (cit. on p. 79).

[90] M. Al-Maolegi and B. Arkok (2014). "An improved Apriori algorithm for association rules". In: *arXiv preprint arXiv:1403.3948*. DOI: 10.5121/IJNLC.2014.3103 (cit. on p. 15).

[91] M. Matijaš, J. Suykens, and S. Krajcar (2013). "Load forecasting using a multivariate meta-learning system". In: *Expert systems with applications,* vol. 40, no. 11, pp. 4427–4437. DOI: 10.1016/j.eswa.2013.01.047 (cit. on pp. 8, 9, 32).

[92] M. Matijaš, J. Suykens, and S. Krajcar (2013). "Load forecasting using a multivariate meta-learning system". In: *Expert systems with applications*, vol. 40, no. 11, pp. 4427–4437. DOI: 10.1016/j.eswa.2013.01.047 (cit. on p. 70).

[93] N. Meade (2000). "Evidence for the selection of forecasting methods". In: *Journal of forecasting,* vol. 19, no. 6, pp. 515–535. DOI: 10.1002/1099-131X(200011)19:6<515::AID-FOR754>3.0.CO;2-7 (cit. on p. 32).

[94] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, and others (2016). "Mllib: Machine learning in apache spark". In: *The Journal of Machine Learning Research,* vol. 17, no. 1, pp. 1235–1241 (cit. on p. 35).

[95] P. Merson and J. Yoder (2020). "Modeling microservices with DDD". In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 7–8. DOI: 10.1109/ICSA-C50368.2020.00010 (cit. on pp. 21, 22).

[96] M. Mikowski and J. Powell (2013). *Single page web applications: JavaScript end-to-end* (cit. on p. 44).

[97] E. Mocanu, P. Nguyen, M. Gibescu, and W. Kling (2016). "Deep learning for estimating building energy consumption". In: *Sustainable Energy, Grids and Networks*, vol. 6, pp. 91–99. DOI: 10.1016/j.segan.2016.02.005 (cit. on pp. 70, 79).

[98] J. Narwariya, P. Malhotra, L. Vig, G. Shroff, and T. Vishnu (2020). "Meta-learning for few-shot time series classification". In: *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pp. 28–36. DOI: 10.1145/3371158.3371162 (cit. on pp. 31, 32).

[99] M. Nehrir, P. Dolan, V. Gerez, and W. Jameson (1995). "Development and validation of a physically-based computer model for predicting winter electric heating loads". In: *IEEE Transactions on Power Systems*, vol. 10, no. 1. DOI: 10.1109/59.373949 (cit. on p. 79).

[100] K. Ng and N. Awang (2018). "Multiple linear regression and regression with time series error models in forecasting PM10 concentrations in Peninsular Malaysia". In: *Environmental monitoring and assessment,* vol. 190, no. 2, pp. 1–11. DOI: 10.1007/s10661-017-6419-z (cit. on p. 55).

[101] A. Ng (2011). "Sparse autoencoder". In: *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19 (cit. on p. 92).

[102] J. Nielsen and R. Molich (1990). "Heuristic evaluation of user interfaces". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 249–256. DOI: 10.1145/97243.97281 (cit. on p. 44).

[103] J. Nielsen (1994). "Enhancing the explanatory power of usability heuristics". In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 152–158. DOI: 10.1145/191666.191729 (cit. on p. 44).

[104] N. Noor, M. Al Bakri Abdullah, A. Yahaya, and N. Ramli (2015). "Comparison of linear interpolation method and mean method to replace the missing values in environmental data set". In: *Materials Science Forum*. Vol. 803, pp. 278–281. DOI: 10.4028/www.scientific.net/MSF.803.278 (cit. on p. 102).

[105] R. Obe and L. Hsu (2017). *PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database* (cit. on p. 52).

[106] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke (2017). "Tensorflow-serving: Flexible, high-performance ml serving". In: *arXiv preprint arXiv:1712.06139*. DOI: 10.48550/arXiv.1712.06139 (cit. on pp. 30, 36).

[107] F. Osisanwo, J. Akinsola, O. Awodele, J. Hinmikaiye, O. Olakanmi, and J. Akinjobi (2017). "Supervised machine learning algorithms: classification and comparison". In: *International Journal of Computer Trends and Technology (IJCTT),* vol. 48, no. 3, pp. 128–138. DOI: 10.14445/22312803/IJCTT-V48P126 (cit. on p. 14).

[108] A. Oussous, F. Benjelloun, A. Lahcen, and S. Belfkih (2018). "Big Data technologies: A survey". In: *Journal of King Saud University-Computer and Information Sciences,* vol. 30, no. 4, pp. 431–448. DOI: 10.1016/j.jksuci.2017.06.001 (cit. on p. 18).

[109] J. Padmanabhan and M. Jose Johnson Premkumar (2015). "Machine Learning in Automatic Speech Recognition: A Survey". In: *IETE Technical Review,* vol. 32, no. 4, pp. 240–251. DOI: 10.1080/02564602.2015.1010611 (cit. on p. 1).

[110] B. Pang, E. Nijkamp, and Y. Wu (2020). "Deep learning with tensorflow: A review". In: *Journal of Educational and Behavioral Statistics,* vol. 45, no. 2, pp. 227–248. DOI: 10.3102/1076998619872761 (cit. on pp. 27, 36).

[111] N. Papahristou and I. Refanidis (2011). "Training neural networks to play backgammon variants using reinforcement learning". In: *Applications of Evolutionary Computation*, pp. 113–122. DOI: 10.1007/978-3-642-20525-5_12 (cit. on p. 16).

[112] R. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj (2016). "Generalized hampel filters". In: *EURASIP Journal on Advances in Signal Processing,* vol. 2016, no. 1, pp. 1–18. DOI: 10.1186/s13634-016-0383-6 (cit. on pp. 75, 86).

[113] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg (2011). "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research*, vol. 12, pp. 2825–2830 (cit. on p. 35).

[114] K. Perera, Z. Aung, and W. Woon (2014). "Machine learning techniques for supporting renewable energy generation and integration: a survey". In: *International Workshop on Data Analytics for Renewable Energy Integration*, pp. 81–96. DOI: 10.1007/978-3-319-13290-7_7 (cit. on p. 1).

[115] C. Persson, P. Bacher, T. Shiga, and H. Madsen (2017). "Multi-site solar power forecasting using gradient boosted regression trees". In: *Solar Energy*, vol. 150, pp. 423–436. DOI: 10.1016/j.solener.2017.04.066 (cit. on p. 55).

[116] J. Pimentel, L. Murta, V. Braganholo, and J. Freire (2019). "A large-scale study about quality and reproducibility of jupyter notebooks". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 507–517. DOI: 10.1109/MSR.2019.00077 (cit. on p. 2).

[117] V. Popovici, W. Chen, B. Gallas, C. Hatzis, W. Shi, F. Samuelson, Y. Nikolsky, M. Tsyganova, A. Ishkin, T. Nikolskaya, and others (2010). "Effect of training-sample size and classification difficulty on the accuracy of genomic predictors". In: *Breast Cancer Research,* vol. 12, no. 1, pp. 1–13. DOI: 10.1186/bcr2468 (cit. on p. 97).

[118] R. Prudêncio and T. Ludermir (2004). "Meta-learning approaches to selecting time series models". In: *Neurocomputing*, vol. 61, pp. 121–137. DOI: 10.1016/j.neucom.2004.03.008 (cit. on pp. 8, 31, 32).

[119] G. Qi, X. Hua, Y. Rui, J. Tang, T. Mei, and H. Zhang (2007). "Correlative multi-label video annotation". In: *Proceedings of the 15th ACM international conference on Multimedia*, pp. 17–26. DOI: 10.1145/1291233.1291245 (cit. on p. 14).

[120] F. Rademacher, J. Sorgalla, and S. Sachweh (2018). "Challenges of domain-driven microservice design: A model-driven perspective". In: *IEEE Software,* vol. 35, no. 3, pp. 36–43. DOI: 10.1109/MS.2018.2141028 (cit. on p. 21).

[121] M. Reif, F. Shafait, and A. Dengel (2012). "Dataset generation for meta-learning". In: *KI-2012: Poster and Demo Track*, pp. 69–73 (cit. on pp. 9, 34, 97).

[122] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel (2014). "Automatic classifier selection for non-experts". In: *Pattern Analysis and Applications*, vol. 17, no. 1, pp. 83–96. DOI: 10.1007/s10044-012-0280-z (cit. on pp. 5, 31, 97).

[123] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. Tenenbaum, H. Larochelle, and R. Zemel (2018). "Meta-learning for semi-supervised few-shot classification". In: *arXiv preprint arXiv:1803.00676*. DOI: 10.48550/arXiv.1803.00676 (cit. on p. 31).

[124] M. Ribeiro, K. Grolinger, and M. Capretz (2015). "Mlaas: Machine learning as a service". In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 896–902. DOI: 10.1109/ICMLA.2015.152 (cit. on p. 29).

[125] J. Rice (1976). "The algorithm selection problem". In: *Advances in computers,* vol. 15, pp. 65–118. DOI: 10.1016/S0065-2458(08)60520-3 (cit. on p. 5).

[126] J. Richens, C. Lee, and S. Johri (2020). "Improving the accuracy of medical diagnosis with causal machine learning". In: *Nature communications*, vol. 11, no. 1, pp. 1–9. DOI: 10.1038/s41467-020-17419-7 (cit. on pp. 1, 14).

[127] J. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren (2014). "Algorithm selection on data streams". In: *International Conference on Discovery Science*. Vol. 8777, pp. 325–336. DOI: 10.1007/978-3-319-11812-3_28 (cit. on pp. 5, 31).

[128] J. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren (2015). "Having a blast: Meta-learning and heterogeneous ensembles for data streams". In: *2015 ieee international conference on data mining*, pp. 1003–1008. DOI: 10.1109/ICDM.2015.55 (cit. on pp. 5, 31).

[129] M. Ringnér (2008). "What is principal component analysis?" In: *Nature biotechnology,* vol. 26, no. 3, pp. 303–304. DOI: 10.1038/nbt0308-303 (cit. on p. 142).

[130] R. Roelofs, V. Shankar, B. Recht, S. Fridovich-Keil, M. Hardt, J. Miller, and L. Schmidt (2019). "A meta-analysis of overfitting in machine learning". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, vol. 32, no. 823, pp. 9179–9189 (cit. on p. 15).

[131] A. Rossi, A. de Leon Ferreira, C. Soares, and B. De Souza (2014). "MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data". In: *Neurocomputing*, vol. 127, pp. 52–64. DOI: 10.1016/j.neucom.2013.05.048 (cit. on p. 38).

[132] M. Rußwurm, S. Wang, M. Körner, and D. Lobell (2020). "Meta-Learning for Few-Shot Land Cover Classification". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 788–796. DOI: 10.1109/CVPRW50498.2020.00108 (cit. on p. 31).

[133] S. Sagiroglu and D. Sinanc (2013). "Big data: A review". In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 42–47. DOI: 10.1109/CTS.2013.6567202 (cit. on p. 18).

[134] S. Salloum, R. Dautov, X. Chen, P. Peng, and J. Huang (2016). "Big data analytics on Apache Spark". In: *International Journal of Data Science and Analytics*, vol. 1, no. 3, pp. 145–164. DOI: 10.1007/s41060-016-0027-9 (cit. on p. 20).

[135] R. Saravanan and P. Sujatha (2018). "A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification". In: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 945–949. DOI: 10.1109/ICCONS.2018.8663155 (cit. on p. 15).

[136] J. Schaad, B. Ramsdell, and S. Turner (2019). "Secure/multipurpose internet mail extensions (S/MIME) version 4.0 message specification". In: *RFC 8551*. DOI: 10.17487/RFC8551 (cit. on p. 23).

[137] S. Schelter, J. Boese, J. Kirschnick, T. Klein, and S. Seufert (2017). "Automatically tracking metadata and provenance of machine learning experiments". In: *Machine Learning Systems Workshop at NIPS*, pp. 27–29 (cit. on p. 30).

[138] B. Scholz-Reiter, M. Kück, and D. Lappe (2014). "Prediction of customer demands for production planning–Automated selection and configuration of suitable prediction methods". In: *CIRP Annals*, vol. 63, no. 1, pp. 417–420. DOI: 10.1016/j.cirp.2014.03.106 (cit. on p. 32).

[139] F. Sebastiani (2002). "Machine learning in automated text categorization". In: *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47. DOI: 10.1145/505282.505283 (cit. on p. 1).

[140] M. Segal (2004). "Machine learning benchmarks and random forest regression". In: *eScholarship Repository. University of California* (cit. on p. 55).

[141] S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2019). "Facilitating and managing machine learning and data analysis tasks in Big Data environments using web and microservice technologies". In: *Proceedings of the 11th International Conference on Management of Digital EcoSystems*, pp. 80–87. DOI: 10.1145/3297662.3365807 (cit. on pp. 42, 44).

[142] S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies". In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems XLV: Special Issue on Data Management and Knowledge Extraction in Digital Ecosystems*, pp. 132–171. DOI: 10.1007/978-3-662-62308-4_6 (cit. on pp. 42, 44).

[143] S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Descriptive Statistics Time-based Meta Features (DSTMF) Constructing a Better Set of Meta Features for Model Selection in Energy Time Series Forecasting". In: *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, pp. 1–6. DOI: 10.1145/3378184.3378221 (cit. on pp. 70, 71, 133, 136).

[144] S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). "Incorporating Unsupervised Deep Learning into Meta Learning for Energy Time Series Forecasting". In: *Proceedings of the Future Technologies Conference*, pp. 326–345. DOI: 10.1007/978-3-030-63128-4_25 (cit. on pp. 70, 78).

[145] S. Shahoud, H. Khalloof, M. Winter, C. Duepmeier, and V. Hagenmeyer (2020). "A Meta Learning Approach for Automating Model Selection in Big Data Environments using Microservice and Container Virtualization Technologies". In: *Proceedings of the 12th International Conference on Management of Digital EcoSystems*, pp. 84–91. DOI: 10.1145/3415958.3433072 (cit. on p. 132).

[146] S. Shahoud, M. Winter, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2021). "An extended Meta Learning Approach for Automating Model Selection in Big Data Environments using Microservice and Container Virtualization Technologies". In: *Internet of Things,* vol. 16, p. 100432. DOI: 10.1016/j.iot.2021.100432 (cit. on pp. 13, 97).

[147] K. Shastry and H. Sanjay (2020). "Machine learning for bioinformatics". In: *Statistical modelling and machine learning principles for bioinformatics techniques, tools, and applications*, pp. 25–39. DOI: 10.1007/978-981-15-2445-5_3 (cit. on p. 14).

[148] C. Shrestha (2016). "A Web Based User Interface for Machine Learning Analysis of Health and Education Data". In: DOI: 10.34917/9112192 (cit. on p. 28).

[149] K. Shvachko, H. Kuang, S. Radia, and R. Chansler (2010). "The hadoop distributed file system". In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10. DOI: 10.1109/MSST.2010.5496972 (cit. on pp. 18, 42).

[150] A. Singh, N. Thakur, and A. Sharma (2016). "A review of supervised machine learning algorithms". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315 (cit. on p. 14).

[151] K. Smith-Miles (2009). "Cross-disciplinary perspectives on meta-learning for algorithm selection". In: *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, pp. 1–25. DOI: 10.1145/1456650.1456656 (cit. on pp. 5, 8, 32, 69, 70, 97).

[152] C. Staff (2016). "React: Facebook's functional turn on writing Javascript". In: *Communications of the ACM,* vol. 59, no. 12, pp. 56–62. DOI: 10.1145/2980991 (cit. on p. 44).

[153] A. Stief, J. Ottewill, and J. Baranowski (2018). "Relief F-based feature ranking and feature selection for monitoring induction motors". In: *2018 23rd international conference on methods & models in automation & robotics (MMAR)*, pp. 171–176. DOI: 10.1109/MMAR.2018.8486097 (cit. on p. 32).

[154] V. Subramanian (2019). "React-Bootstrap". In: *Pro MERN Stack*, pp. 315–376. DOI: 10.1007/978-1-4842-4391-6_11 (cit. on p. 44).

[155] S. Taieb, G. Bontempi, A. Atiya, and A. Sorjamaa (2012). "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition". In: *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083. DOI: 10.1016/j.eswa.2012.01.039 (cit. on p. 114).

[156] T. Talagala, R. Hyndman, G. Athanasopoulos, and others (2018). "Meta-learning how to forecast time series". In: *Monash Econometrics and Business Statistics Working Papers* (cit. on pp. 9, 34, 97, 108).

[157] R Core Team (2014). "A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna". In: *Austria2014* (cit. on pp. 35, 77).

[158] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown (2013). "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855. DOI: 10.1145/2487575.2487629 (cit. on pp. 10, 32).

[159] R. Tibshirani, G. Walther, and T. Hastie. "Estimating the number of clusters in a data set via the gap statistic". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423. DOI: 10.1111/1467-9868.00293 (cit. on p. 82).

[160] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas (2008). "Multi-label classification of music into emotions." In: *ISMIR*. Vol. 8, pp. 325–330 (cit. on p. 14).

[161] G. Tso and K. Yau (2007). "Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks". In: *Energy*, vol. 32, no. 9, pp. 1761–1768. DOI: 10.1016/j.energy.2006.11.010 (cit. on p. 55).

[162] J. Van Engelen and H. Hoos (2020). "A survey on semi-supervised learning". In: *Machine Learning*, vol. 109, no. 2, pp. 373–440. DOI: 10.1007/s10994-019-05855-6 (cit. on p. 16).

[163] J. Vanschoren (2018). "Meta-learning: A survey". In: *arXiv preprint arXiv:1810.03548*. DOI: 10.48550/arXiv.1810.03548 (cit. on p. 31).

[164] J. Vanschoren (2019). "Meta-learning". In: *Automated Machine Learning*, pp. 35–61. DOI: 10.1007/978-3-030-05318-5_2 (cit. on p. 31).

[165] M. Vartak, J. F. da Trindade, S. Madden, and M. Zaharia (2018). "Mistique: A system to store and query model intermediates for model diagnosis". In: *Proceedings of the 2018 International Conference on Management of Data*, pp. 1285–1300. DOI: 10.1145/3183713.3196934 (cit. on pp. 30, 36).

[166] M. Vartak, H. Subramanyam, W. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia (2016). "ModelDB: a system for machine learning model management". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pp. 1–3. DOI: 10.1145/2939502.2939516 (cit. on pp. 3, 27, 29, 36).

[167] V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, and S. Seth (2013). "Apache hadoop yarn: Yet another resource negotiator". In: *Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1–16. DOI: 10.1145/2523616.2523633 (cit. on pp. 20, 42).

[168] A. Venkatachalam and J. Sohl (1999). "An intelligent model selection and forecasting system". In: *Journal of Forecasting*, vol. 18, no. 3, pp. 167–180. DOI: 10.1002/(SICI)1099-131X(199905)18:3<167::AID-FOR715>3.0.CO;2-F (cit. on p. 32).

[169] C. Wan, J. Zhao, Y. Song, Z. Xu, J. Lin, and Z. Hu (2015). "Photovoltaic and solar power forecasting for smart grid energy management". In: *CSEE Journal of Power and Energy Systems*, vol. 1, no. 4, pp. 38–46. DOI: 10.17775/CSEEJPES.2015.00046 (cit. on pp. 76, 113).

[170] B. Wang, Y. Li, W. Ming, and S. Wang (2020). "Deep reinforcement learning method for demand response management of interruptible load". In: *IEEE Transactions on Smart Grid,* vol. 11, no. 4, pp. 3146–3155. DOI: 10.1109/TSG.2020.2967430 (cit. on pp. 1, 14, 70).

[171] J. Wang (2021). "Meta-learning in natural and artificial intelligence". In: *Current Opinion in Behavioral Sciences*, vol. 38, pp. 90–95. DOI: 10.1016/j.cobeha.2021.01.002 (cit. on p. 31).

[172] A. Widodo and I. Budi (2013). "Model selection using dimensionality reduction of time series characteristics". In: *International Symposium on Forecasting, Seoul, South Korea*, pp. 57–118 (cit. on p. 31).

[173] D. Wolpert (1996). "The lack of a priori distinctions between learning algorithms". In: *Neural computation*, vol. 8, no. 7, pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341 (cit. on pp. 4, 131).

[174] B. Wu, S. Lyu, B. Hu, and Q. Ji (2015). "Multi-label learning with missing labels for image annotation and facial action unit recognition". In: *Pattern Recognition,* vol. 48, no. 7, pp. 2279–2289. DOI: 10.1016/j.patcog.2015.01.022 (cit. on p. 14).

[175] J. Wu, W. Xiong, and W. Wang (2019). "Learning to learn and predict: a meta-learning approach for multi-label classification". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4354–4364. DOI: 10.18653/v1/D19-1444 (cit. on p. 31).

[176] B. Xian, X. Zhang, H. Zhang, and X. Gu (2021). "Robust Adaptive Control for a Small Unmanned Helicopter Using Reinforcement Learning". In: *IEEE Transactions on Neural Networks and Learning Systems.* DOI: 10.1109/TNNLS.2021.3085767 (cit. on p. 16).

[177] D. Xu and Y. Tian (2015). "A comprehensive survey of clustering algorithms". In: *Annals of Data Science,* vol. 2, no. 2, pp. 165–193. DOI: 10.1007/s40745-015-0040-1 (cit. on p. 82).

[178] L. Xu, S. Wang, and R. Tang (2019). "Probabilistic load forecasting for buildings considering weather forecasting uncertainty and uncertain peak load". In: *Applied Energy*, vol. 237, pp. 180–195. DOI: 10.1016/j.apenergy.2019.01.022 (cit. on pp. 1, 14).

[179] A. Yang, W. Li, and X. Yang (2019). "Short-term electricity load forecasting based on feature selection and Least Squares Support Vector Machines". In: *Knowledge-Based Systems*, vol. 163, pp. 159–173. DOI: 10.1016/j.knosys.2018.08.027 (cit. on pp. 1, 14, 70).

[180] X. Yang, Z. Song, I. King, and Z. Xu (2022). "A survey on deep semi-supervised learning". In: *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–20. DOI: 10.1109/TKDE.2022.3220219 (cit. on p. 16).

[181] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma (2020). "Rethinking bias-variance trade-off for generalization of neural networks". In: *International Conference on Machine Learning*. Vol. 119, pp. 10767–10777 (cit. on p. 15).

[182] Y. Yao, Z. Xiao, B. Wang, B. Viswanath, H. Zheng, and B. Zhao (2017). "Complexity vs. performance: empirical analysis of machine learning as a service". In: *Proceedings of the 2017 Internet Measurement Conference*, pp. 384–397. DOI: 10.1145/3131365.3131372 (cit. on p. 29).

[183] X. Ying (2019). "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. 2, p. 022022. DOI: 10.1088/1742-6596/1168/2/022022 (cit. on p. 15).

[184] C. Yu, P. Mirowski, and T. Ho (2016). "A sparse coding approach to household electricity demand forecasting in smart grids". In: *IEEE Transactions on Smart Grid,* vol. 8, no. 2, pp. 738–748. DOI: 10.1109/TSG.2015.2513900 (cit. on p. 79).

[185] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, and M. Parkhe (2018). "Accelerating the machine learning lifecycle with MLflow". In: *IEEE Data Eng. Bull,* vol. 41, no. 4, pp. 39–45 (cit. on pp. 3, 30, 36).

[186] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. Franklin, S. Shenker, and I. Stoica (2012). "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 15–28 (cit. on p. 58).

[187] M. Zaharia, R. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. Franklin, and others (2016). "Apache spark: a unified engine for big data processing". In: *Communications of the ACM,* vol. 59, no. 11, pp. 56–65. DOI: 10.1145/2934664 (cit. on pp. 27, 36, 42).

[188] C. Zhang, S. Kuppannagari, R. Kannan, and V. Prasanna (2018). "Generative adversarial network for synthetic time series data generation in smart grids". In: *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6. DOI: 10.1109/SmartGridComm.2018.8587464 (cit. on pp. 9, 34, 98).

[189] H. Zhang, L. Zhang, and Y. Jiang (2019). "Overfitting and underfitting analysis for deep learning based end-to-end communication systems". In: *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6. DOI: 10.1109/WCSP.2019.8927876 (cit. on p. 15).

[190] J. Zhang, Y. Wei, D. Li, Z. Tan, and J. Zhou (2018). "Short term electricity load forecasting using a hybrid model". In: *Energy*, vol. 158, pp. 774–781. DOI: 10.1016/j.energy.2018.06.012 (cit. on p. 70).

[191] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng (2019). "Meta-gnn: On few-shot node classification in graph meta-learning". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2357–2360. DOI: 10.1145/3357384.3358106 (cit. on p. 31).

A. List of Publications

Journal Articles

1. S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). “Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies”. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XLV: Special Issue on Data Management and Knowledge Extraction in Digital Ecosystems, pp. 132–171. doi: 10.1007/978-3-662-62308-4_6.

2. S. Shahoud, M. Winter, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2021). “An extended Meta Learning Approach for Automating Model Selection in Big Data Environments using Microservice and Container Virtualization Technologies”. In: Internet of Things, vol. 16, p. 100432. doi: 10.1016/j.iot.2021.100432.

Conference Articles

1. S. Shahoud, S. Gunnarsdottir, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2019). “Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies”. In: Proceedings of the 11th International Conference on Management of Digital EcoSystems, pp. 80–87. doi: 10.1145/3297662.3365807.

2. S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). “Descriptive Statistics Time-based Meta Features (DSTMF) Constructing a Better Set of Meta Features for Model Selection in Energy Time Series Forecasting”. In: Proceedings of the 3rd International Conference on Applications of Intelligent Systems, pp. 1–6. doi: 10.1145/3378184.3378221.

3. S. Shahoud, H. Khalloof, C. Duepmeier, and V. Hagenmeyer (2020). “Incorporating Unsupervised Deep Learning into Meta Learning for Energy Time Series Forecasting”. In: Proceedings of the Future Technologies Conference, pp. 326–345. doi: 10.1007/978-3-030-63128-4_25.

4. S. Shahoud, H. Khalloof, M. Winter, C. Duepmeier, and V. Hagenmeyer (2020). “A Meta Learning Approach forAutomating Model Selection in Big Data Environments Using Microservice and Container Virtualization Technologies”. In: Proceedings of the 12th International Conference on Management of Digital EcoSystems, pp. 84–91. doi: 10.1145/3415958.3433072.