Universität für Bodenkultur, Wien

University of Natural Resources and Life Sciences, Vienna

Department für Wirtschafts- und Sozialwissenschaften

Department of Economics and Social Sciences

Evaluation of Machine Learning-Based Storage Control Algorithms for the EEX Day-Ahead Electricity Market

Master's Thesis

Field of Study:

Masterstudium Umwelt- und Bioressourcenmanagement
(Master's Program Environment and Bio-Resources Management)

Supervisor:

Johannes Schmidt, Ass.Prof. Dipl.-Ing. Dr. (BOKU, Vienna)

Author:

Lukas Zwieb, BSc (1040726)

April 18, 2018

Statutory Declaration

I declare that I have developed and written the enclosed Master thesis titled 'Evaluation of Machine Learning based Storage Control Algorithms for the EEX Day-Ahead Electricity Market' completely by myself and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly indicated. The Master Thesis was not used in the same or in a similar version to achieve an academic grading or has been published elsewhere.

Vienna, February 15, 2018 Lukas Zwieb

Acknowledgments

I want to express my gratitude for the valuable advice and support provided by my supervisor Dr. Johannes Schmidt during the work on this thesis. I also want to thank my family, friends and colleagues who directly and indirectly supported me. I especially thank my girlfriend Annemarie for all her support.

Abstract

In context of climate change mitigation, power supply systems around the world are changing towards sustainable sources of power supply. This change is becoming a key challenge for managing the electrical grid. Energy storage systems (ESS) can increase power system flexibility and efficiency and facilitate integration of variable renewable energy. ESS deployment will remain limited if its operation is not profitable. Automated, optimized operation of the ESS can help to increase profits from temporal arbitrage. Machine learning algorithms (ML) are one option to provide the operational logic behind an automated trading process.

This thesis uses a deterministic linear optimization model (LOM) with full forecast information to determine the best possible (but in practice not achievable) trading strategy for a lithium Ion based EES on the German day-ahead market (2010-2014). The results are used as a benchmark to train different ML-classifiers to mimic the 'optimal behavior' of the LOM results. The final result is a model to evaluate the classifiers performances under simulated market conditions for the year 2015. The evaluation shows that *Neuronal Networks*, *K-nearest neighbors*, and *random forest* reach approximately 80% of the LOM benchmark profit. Although the results of the LOM show that, due to high investment costs, the operation of an ESS is not yet economically attainable, it also shows that ML based applications are suited for future automated ESS control.

Abstract

Im Kontext von Klimawandelsmitigation wandeln sich Energiesysteme weltweit in Richtung nachhaltiger Energieversorgung. Diese Veränderung entwickelt sich zu einer wesentlichen Herausforderung für das Energienetz. Energiespeichersysteme können die Flexibilität und Effizienz des Energiesystems erhöhen und ermöglichen die Integration variabler Anteile erneuerbarer Energien. Die Nutzung von Energiespeichersystemen ist jedoch limitiert, solange die Systeme nicht profitabel sind. Ein automatisierter Betrieb der Systeme kann zu einer Senkung der Betriebskosten und so zu einer Steigerung der Profitabilität beitragen. Maschinelle Lernalgorithmen (ML) sind dabei eine Möglichkeit, die einem automatisierten Handelsprozess zugrundeliegende operative Logik bereitzustellen.

Diese Arbeit nutzt daher ein deterministisches lineares Optimierungsmodell mit voller Information über zukünftige Preise, um den Handel eines Lithium-Ionen basierten Energiespeichersystems am deutschen Energiemarkt (2010-2014) zu optimieren. Die gewonnenen Ergebnisse werden als Benchmark verwendet, um ML-Algorithmen auf die Nachahmung des vom linearen Optimierungsmodell vorgegebenen „optimalen Verhaltens" zu trainieren. Das Endergebnis ist ein Modell zur Evaluierung der Performance des Algorithmus unter den Marktbedingungen von 2015. Die Evaluierung zeigt, dass *Neuronale Netzwerke*, *K-Nearest Neighbors* und *Random Forest* zu etwa 80 % den vom linearen Optimierungsmodell festgesetzten Benchmark-Profit erreichen. Obwohl die Ergebnisse zeigen, dass der Betrieb eines Energiespeichersystems aufgrund der hohen Investitionskosten zum jetzigen Zeitpunkt nicht profitabel ist, zeigen sie auch, dass ML-basierte Ansätze für die Steuerung von Energiespeichersystemen zukünftig nutzbar sind.

Table of Content

i. List of Figures

ii. List of Tables

iii. List of Abbreviations

API	Application Programming Interface
CAES	Compressed Air Energy Storage
COSAS	Classified OSA-Signals
DSM	Demand Side Management
E	Energy [J]
E2P	Energy to Power Ratio
EES	Electrical Energy Storage
EEX	European Energy Exchange
EM	Electricity Market
ESS	Energy Storage System
ENTSOE	European Network of Transmission System Operators for Electricity
IRR	Internal Rate of Return
KNN	L-Nearest Neighbors
LC	Load Cycle
LC	Load Cycles
Li-ion	Lithium Ion Battery
LP	Linear Programming
LPM	Linear Programming Model
LR	Logistic Regression
ML	Machine Learning
MLA	Machine Learning Algorithm
MLP	Multilayer Perceptron
OSA	Optimal Storage Action
P	Power [W]
P2E	Power to Energy Ratio
PCA	Principal Component Analysis
PHPP	Pumped Hydro Power Plant
PP	Power Plant
rbf	Radial Bias Function
RF	Random Forest
RM	Rolling Mean / Moving Average
SDR	Self-Discharge Rate
SL	Storage Logic
SVC	Support Vector Classifier
VRE	Volatile Renewable Generation

1 Introduction

To mitigate the effects of climate change and reduce the dependency on fossil resources, the energy supply system must be modified profoundly. This includes transport and heat supply as well as electricity generation. Fossil fuels must be replaced by sustainable, emission free energy sources. This process includes the interconnection of different sectors and countries, fundamental efficiency gains and a general flexibilization of the whole energy supply system [1]. The necessity of a collaborative global effort found its acknowledgement during the UN Climate Change Conference[1] which took place 2015 in Paris.

Germany is seen as a front runner of this transformation process, as it implemented strong policies towards the energy transformation and has high economic potential [2]. Based on the goals of the 'renewable energy act' (ger.: Gesetz für den Ausbau erneuerbarer Energien (Kurztitel: Erneuerbare-Energien-Gesetz, EEG 2017[2]), which accurately outlines the shift from a carbon-nuclear into a wind-solar based energy supply system, the German 'energy transformation' (ger.: Energiewende) is already well underway. A more detailed view on the electricity fuel mix itself reveals the necessity to substitute large shares of coal, lignite, and nuclear fueled generation with renewable sources like wind and solar power, hydropower, and biomass (Figure 1-1). Other power plants (PP) like geothermal PP, tidal PP or fusion reactors are either limited to niche applications or not yet technically mature [3], [4]. Germany's geological potential for hydropower plants is largely exploited and the limits to bulk usage of biomass based energy is still controversially discussed within the scientific community [4]–[6]. Therefore studies find the most plausible alternative for the realization of large scale renewable energy (RE) integration in Germany within volatile solar and wind power [4], [5], [7], [8].

1 21st Conference of the Parties to the United Nations Framework Convention on Climate Change (UNFCCC).

2 Erneuerbare-Energien-Gesetz vom 21. Juli 2014 (BGBl. I S. 1066), zuletzt geändert durch Artikel 1 des Gesetzes vom 17. Juli 2017 (BGBl. I S. 2532).

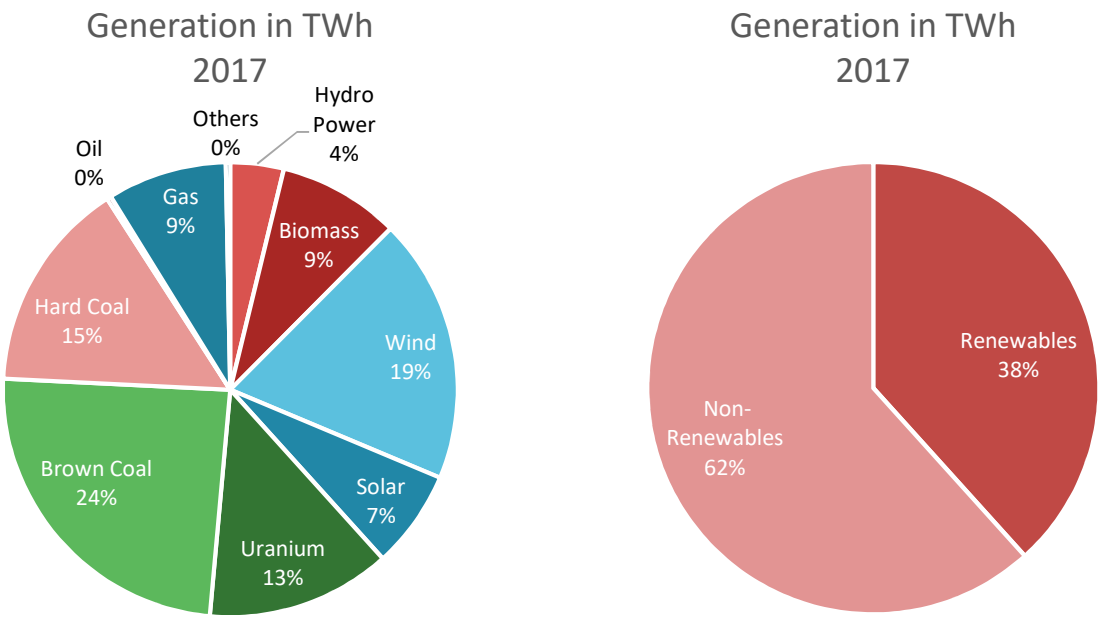


Figure 1-1: Fuel mix in the German energy system for the year 2017 [9].

The German energy supply system is organized as a liberalized market and is part of the integrated European Energy market. The electricity market (EM) in general differs from typical commodity markets in two main aspects. First, power supply systems require a balance between supply and demand at any point in time. A mismatch leads to reduced quality and ultimately blackouts [3], [4]. Secondly, electricity is – by its nature - difficult to store (e.g. keep in stock at customers). Simultaneously, the demand is not (yet) controllable and volatile on sub-hourly, hourly, daily, weekly, and seasonal levels. Consequently, the generation must meet the consumption at all times [10]–[13].

As of today, thermal power plants characterize the European power plant fleet. Their combustion, and therefore the power output, is adjustable. Demand fluctuations and forecasting errors can be met by adjusting the power of multiple power plants gradually and additional capacities can be added with short ramping times. Solar and wind power as the primary sources of renewable electricity on the other hand are fluctuating and volatile, as are solar radiation and wind speeds [11]. At first glance this appears to be incompatible with the requirements of a balanced power grid. However, numerous studies found no fundamental technical limitations for very high shares of renewable energy in the gross final energy consumption under the assumption of additional flexibility measures [4], [6], [10], [14]. Consequently, the substitution of a thermal power plant fleet (e.g. fossil/nuclear fueled) requires additional means of reactive flexibility and load levelling.

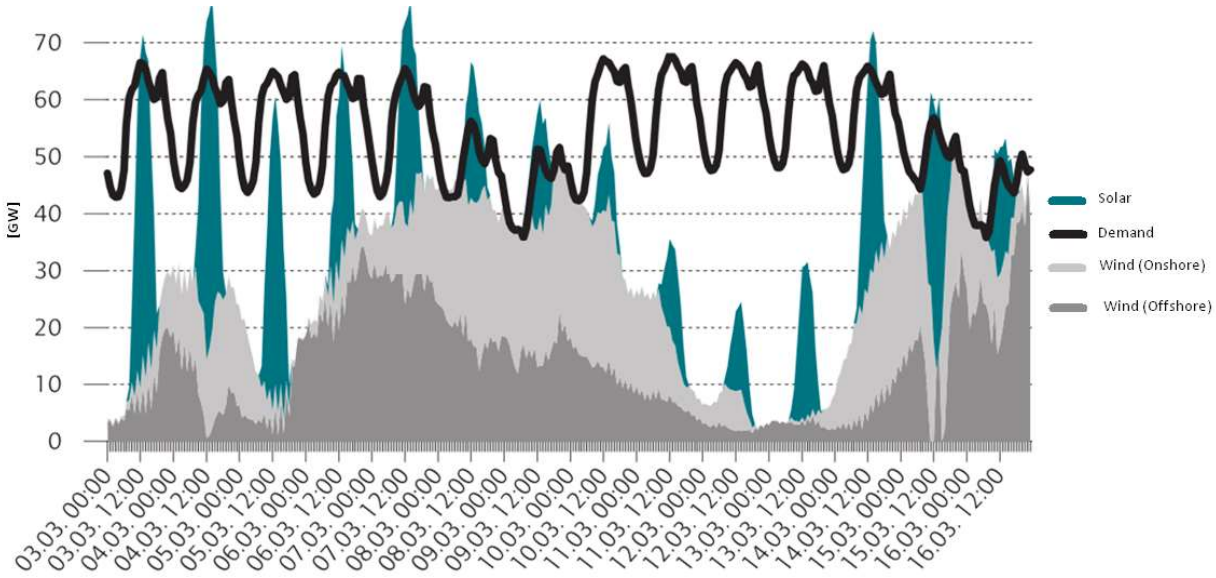


Figure 1-2: Two weeks of a simulation for an 80% renewable energy scenario based on the year 2013 by *Schill* and *Zerrahn* [6].

Figure 1-2 shows the calculations of *Schill* and *Zerrahn* [6] for an 80 % renewable scenario based on the weather and demand data for the year 2013. The simulation illustrates the fluctuations of the volatile residual loads to be handled by energy supply systems. According to *Sterner* and *Stadler*, the technical alternatives capable of providing this flexibility are already available [4]. *Sterner* and *Stadler* differentiate between four different technical approaches to describe these flexibility measures:

- **Electrical energy storages** (EES) increase temporal flexibility. During times of surplus generation (e.g. during night) high volatile renewable generation (VRE) energy is stored and later fed back into the grid during times of increased demand (e.g. windless periods).
- **Grid development** is a mean of spatial flexibility. A developed integrated grid allows a better distribution of local fluctuations. Investments in the infrastructure and controllability improve the resilience of the grid and therefore reduce the demand for flexibility. However, the dispatchable capacities are also limited by the residual load / market situation of the trading partners' power supply system. Only if conditions for a trade are met by both sides this capacity can be used as a flexibility option.
- **Increased flexibility** of the PP-fleet and **reduction of 'must run units'** reduce temporal flexibility demand. Larger power gradients of the remaining thermal PP can meet unexpected fluctuation by quickly adjusting their power. Wind and solar farms can curtail their power generation to provide additional power if necessary.
- **Demand side management** (DSM) is a measure of temporal load shift. In contrast to ESS DSM allows a reduction of the consumption during large positive flexibility demand. This reduction is compensated later (or earlier) during low flexibility demand. Popular technologies enabling DSM are heating and cooling, electric vehicles, or energy intensive industrial processes.

These measures suit different forms of flexibility demand and are not interchangeable. Also, there is a competition between these technologies in terms of economic performance as well as social and ecological acceptance. According to *Nicolosi* [10], this competition can be met by means of the liberalized market, which will lead to an optimal mixture of different flexibility options. *Stadler* and *Sterner* assume that there is a sequence of 'next best' flexibility options determined by economic ecologic and social aspects [4]. However, both agree that EES will play a fundamental role within the future EM.

Although the debate about the EES' role within the future energy supply system is still underway, EES are already providing a bundle of different services, such as avoiding costly interconnecting infrastructure and enabling emission reductions [15]. Apart from this, simulations show that there is a growing demand of EES as the share of VRE within the energy mix rises due to the natural fluctuations of RE sources [5], [7], [16]. Eventually this means that an ongoing expansion of VRE is pushing the demand for EES.

A key factor for the large-scale implementation of EES is its economic performance. As EES can provide multiple services, multiple revenue streams must be considered while evaluating their profitability. The price for electricity at the EM is moving periodically, i.e. it follows daily and weekly cycles. This allows profitable arbitrage trading, but the optimal scheduling of charging and discharging processes is a difficult problem, as the exact levels of future prices are unknown. The optimal storage schedule for past price time series can be optimized with a deterministic linear program. This approach requires a complete set of price levels of the assessment period (e.g. perfect foresight). The result is a schedule of optimal storage decisions, yielding the maximal profit (see chapter 3.2.1) Such models are used to assess the economic potential and the value of EES within a market respectively a power supply system (i.e. micro grids). However, under realistic operation conditions there is only limited information about future price levels available [17], [18], which makes the application of deterministic linear optimization models impossible. Hence, optimizing the storage process during operation requires models capable of providing decisions under uncertainty. EES management is an interdisciplinary task at the intersection of electronic engineering, weather and price forecasting, and probability theory. Thus, methods from different disciplines (e.g. engineering, statistics) are necessary for developing algorithms operating the EES. Naive approaches are for example the net power balance algorithms where the storage is used as a buffer in order to increase self-consumption [19] or the 'back to back' strategy, where the optimal storage schedule of a bygone period is mirrored [18]. Prominent advanced techniques include approaches based on autoregression (AR), moving average (MA), auto regression and moving average (ARMA), fuzzy logic or wavetable analysis[19], [20]. While linear optimization delivers optimal solutions and so represents an upper boundary on the profitability of storage schedules, latter algorithms are prone to sub optimal decisions.

The patterns appearing at the EM could be recognized by data driven algorithms which could then trade profitably on the market independently. Additionally, special requirements of different storage types, such as death of discharge, can easily be implemented. This could reduce the costs of operation while simultaneously outperform static strategies such as 'static and moving average arbitrage' or 'back to back arbitrage' [15], [18]. The objective of my thesis is the evaluation of the feasibility of EES being operated by automated algorithms. This kind of algorithms, which are capable of 'picking up' patterns from data sets, are popularly called 'machine learning algorithms' (MLA). Machine learning (ML) in this case describes the 'machine' learning from a given set of data rather than following strict programmatic rules.[3] The novel approach of using MLA to estimate optimal storage behavior shall help to reduce the gap between the optimal results of linear programmed results and realistic results under the consideration of forecast uncertainties by increasing the realizable revenue of arbitrage trading. To achieve this, I test different MLA for their ability to learn 'how to trade profitably at the EM' and compare them with typical state of the art strategies of storage control. The performance of different MLAs and other trading strategies is evaluated within a testing framework simulating real market conditions.

The second chapter (2 – Technical and Economic Framework), briefly summarizes the most important technical characteristics of EES, the particularities of the EM, and the data used. The third chapter (3 – Evaluation Process) describes and analyzes the market data and sets up the testing framework. It describes the formulation of the Linear Programming Model (LPM) used as reference scenario, the MLAs process, and the setup of the optimization framework. The fourth chapter (4 – Classification Algorithms) examines the inner workings of different MLAs. The fifth chapter (5 – Results), presents and analyzes the results of the evaluation. Chapter 6 discusses and concludes the results. The Appendix provides additional calculations as well as the code of the models. The complete code is also available at: https://github.com/zwiebo/Evaluation-of-Machine-Learning-Based-Storage-Control-Algorithms-for-the-Electricity-Market

2 Technical and Economic Framework

2.1 Storing Electricity

An electricity storage is a technical unit to store (feed-in or charge), conserve, and withdraw (feed-out or discharge) electricity. However, since electricity itself is not storable, this process involves the transformation of electricity into other forms while accepting transformation losses (e.g. electrical potential difference energy into gravitational potential energy). Storing and withdrawing

3 Machine learning is a research field at the intersection of artificial intelligence, computer science and statistics. Today MLA have various areas of application and are successfully used in medicine for automated diagnostics, in finance for fraud detection, as spam filer or as recommendation systems. See for example https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/ (18.04.2018).

energy/electricity allows a temporal re-allocation of electricity supply. Thus, it can help to guarantee balanced electricity markets [4].

2.2 EES Physical Parameters

According to *Stadler* and *Sterner* and *Fuchs* et al. there are several important physical parameters of storage facilities [6], [4], [7]. These parameters vary between the different types of EES and determine their different economic performances under different market conditions. Consequently, different initial conditions will lead to different optimal EES-types. Figure 2-1 shows different types of EES depending on their typical discharging time and storage capacity.

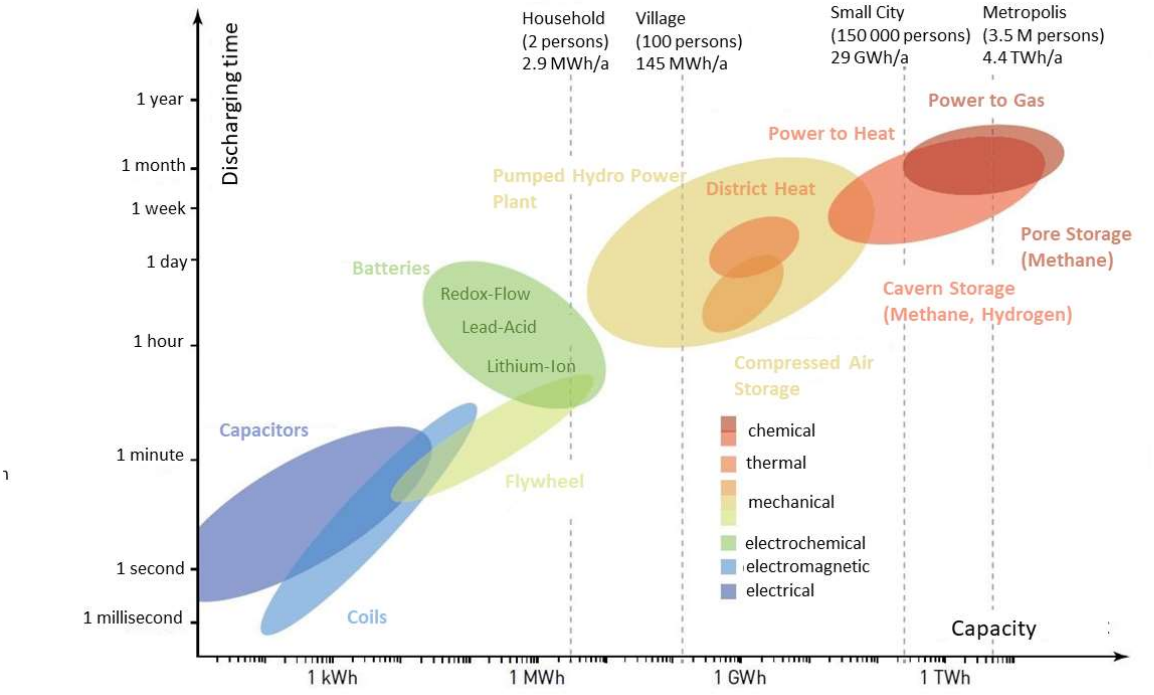


Figure 2-1: Power to energy ratio in Germany realizable storage projects (translated from TH Regensburg FENES, 2013).

Short term storages store energy from nanoseconds up to one day. At the power market, most short-term storages have an energy to power ratio of ≤ 10 (see chapter 2.2.3). The main purpose of these storages is to assure the power quality (i.e. frequency control). The frequent use leads to increased requirements on load cycle efficiency and endurance. Popular examples for short term storages are flywheels, capacitors, and batteries [4].

Long term storages can store energy over several days, weeks or even years, thus temporal fluctuations like seasonal demand or supply oscillations can be compensated. Long term storages are characterized by low self-discharge rates and low power-to-energy (P2E) capacity ratios. Today these large PHPP are limited to mountain regions. Other storages which are based on chemical storage mediums such as in power-to-gas facilities depend on tanks or caverns [4].

Different EES can be compared by using specific parameters which are summarized in Table 1.

Table 1: Summary of the most important physical values according to *Stadler* and *Sterner* (2013) [6].

Parameter	Symbol	Description	Unit
Power	**P**	Charging/discharging power per time	W
Storage Capacity	**C**	Usable storage capacity	J, Wh
Power to Energy ratio	**P2E**	Ratio between Power and Energy	DLU
Efficiency	**η**	Efficiency of the storing process (includes feed in and feed out)	DLU[4] (%)
Self-discharge rate	**ρ**	Relative amount of over time lost energy. (due to self-discharge)	% / time
Load Cycle	**L_c**	Combination of a complete charging and discharging cycle	DLU
Lifetime	**L**	maximal deployable time	Years

2.2.1 Power

The power (P) (in Joule per second) as a physical value describes the rate of energy transfer per time unit. The power (demand and supply) is a crucial parameter determining supply quality. In context of EES the power is separated into charging power (P_{in}) and discharging power (P_{out}). P_{in} and P_{out} must not have the same magnitude. For example, PHPP pumps and generators often have differently dimensioned pumps and generators.[5] This leads to diverging P_{in} and P_{out}, as shown in equation 3-1.

$$P_{\text{in}}, P_{\text{out}} = \frac{dW_{\text{in}},\ dW_{out}}{dt}$$

Eq. 2-1(Dis)charging power.

2.2.2 Storage Capacity

The storage capacity (E_{max}) in Watt-hours [Wh] is the amount of electricity that can be fed in or out of an EES. E_{in} is the integral of the feed in power (P_{in}) over the feed in time t_{in}. E_{out} is the integral of the feed out power (P_{out}) over the feed out time (t_{out}). If *P* is constant, *E* is the product of *P* and *t*. EES often do not exhaust the full potential of their capacity. The amount of energy fed in or out during a partial (dis)charging process can be calculated accordingly. The duration of the (dis)charging processes can be calculated by transforming Eq. 2-2.:

4 Dimensionless unit.

5 Pumps often are not directly controllable while the generators are.

$$C_{\text{in/out}} = \int_{t1}^{t2} P_{\text{in/out}}(t) * dt$$

Eq. 2-2 Calculation of the storage capacity.

2.2.3 Power to Energy Ratio

The relation of energy-to-power (E2P) results in specific charging and discharging times. The E2P is the reciprocal value of the maximum (dis)charging time. This metric is a popular mean to categorize EES. Although there are no explicit delimitations between the categories, it is helpful for finding the according EES for the deliberate use case.

2.2.4 Efficiency

Another metric to compare EES is the efficiency. The energy conversion efficiency (η) is a dimensionless unit. It describes the ratio between the amount of energy [Wh] fed into the EES-system and the amount of energy [Wh] fed out of the system, as shown in Eq. 2-3.

$$\eta = \frac{E_{\text{out}}}{E_{\text{in}}}$$

Eq. 2-3 Calculation of the efficiency.

The efficiency of a complete load cycle (combination of charge and discharge) consists of three sub processes.

- charging efficiency (conversion efficiency) (η_{in})
- efficiency over the storing period (storing efficiency / self-discharge rate) (η_{store} / SDR)
- discharging efficiency (conversion efficiency) (η_{out})

Multiplied, the single efficiencies result in the total efficiency of a load cycle (η_{total}).
The efficiency can also be described as a cost-benefit ratio. The costs are described by the electricity fed into the storage system times the price at the point in time. The profits are determined by the future electricity price. To operate an EES economically, the storing-price must be smaller than the withdrawing-price times the total efficiency, as shown in Eq. 2-4.

$$p_{in} <= p_{out} * \eta_{total}$$

Eq. 2-4 Calculation of the minimal price (p_{out}) to trade profitably [4].

2.2.5 Self-Discharge Rate

The EES loses energy over time at a certain rate, which is why the storing efficiency will always be smaller than 1 (100 %). The self-discharge-rate (SDR) or 'parasitic loss' describes the proportion of the stored energy (E_{stored}), which is lost to the environment over a certain period (E_{loss}) [3], [4], as shown in Eq. 2-5:

$$SDR = \frac{\int (P_{loss} * dt)}{E_{stored}} = \frac{E_{loss}}{E_{stored}}$$

Eq. 2-5 Calculation of the selfdicharge rate (SDR).

2.2.6 Load Cycles

A load cycle (L_c) describes one cycle of full charging and discharging. Many degradation processes are depending on the load-cycle. The load cycle is therefore an important metric to estimate the lifetime of an EES under certain operating conditions. The calculation is shown in Eq. 2-6:

$$L_c(t) = \frac{\int_0^t P_{in}\,(t)}{C}$$

Eq. 2-6 Calculation of the load cycles for the period t.

However, in this thesis a simplified approach is used to calculate the number of load cycles. It neglects non-linear relations between state of charge and material fatigue. The number of load cycles is calculated by dividing the sum of P_{in} by the capacity (compare [12]).

2.2.7 Lifetime

The lifetime of an EES represents the period where its performance satisfies certain criteria. The lifetime usually includes not only the temporal erosion effects on the material, but also typical usage forms. In general, two values determine the life time of an EES: the calendar life in years (L_y) and the maximum load cycles (L_c). Average lifetimes vary depending on the kind of EES and the use case. Batteries for example have an expected lifetime of 5 to 20 years while the lifetime of pumped storage power plants can reach up to 80 years and more. While some ESS like flywheels show no quantifiable degradation per load cycle others, such as lead acid batteries, have limited load cycles of about 2000.[6] The specific lifetime ($L_{(Nc)}$) in years is the minimum of maximal load cycles per year (Nc) in years and the maximal calendar life in years (Ly). The limiting factor which is reached first determines the maximal lifetime of the EES [21]. The respective formula is shown in Eq. 2-7:

$$L(N_c) = \min\left\{\frac{L_c}{N_c}, L_y\right\}$$

Eq. 2-7 Calculation of the lifetime (L_{Nc}).

6 Depending on DOD.

2.3 EES Options

Like other flexibility options, EES have multiple functions and applications within the power supply system. EES can be used to trade at wholesale and control energy markets, they can be operated to optimize the internal consumption of private PV systems, or they can serve as backup or black start reserve. In this context, they provide services to the market which can be monetized.
Fuchs et al. and *Schill*, *Diekmann* and *Zerrahn* describe the essential services an EES can provide within the wholesale energy market [6], [7]:

- **Ancillary services**:

Supplying operation reserve (frequency response reserve and non-spinning reserve) as well as frequency and voltage control to assure the continual flow of electricity. This also includes the black start and re-dispatch abilities of most EES.

- **Peak-Shaving**:

EES have the potential to reduce the peak load (e.g. the maximal power of a supply system) by shifting it to lower demand periods. This reduces the demand for seldomly used high demand generators. The peak-shaving market is a power market [kW].

- **Arbitrage trade**:

Arbitrage takes advantage of temporally varying price levels on the electricity market. It provides a load levelling service to the energy market, i.e. buying and storing energy when electricity prices are low and then selling and discharging the energy back to the grid when prices are high. Contrary to peak-shaving, the arbitrage market is an energy market [kWh] [21]. Potentially, an EES can fulfill all three functions at the 'same' time. Nevertheless, task-optimized EES types have evolved. This means that different services require different characteristics (e.g. E2P ratio, maximal lifecycles) [7]. Stochastic valuations of EES show that a co-optimization of all of the above mentioned business cases yields the best performance [22], [23].

2.4 Storage Selection

Figure 2-1 emphasizes the diversity of different EES types. The arbitrage revenue of an EES depends on the round-trip efficiency and self-discharge of the device as well as on its E2P ratio. For short-term arbitrage trade, the optimal E2P ratio lies between 1 and 14 hours due to the diurnal periodicity of the electric prices [4]. High roundtrip efficiencies and low SDR further increase the profitability. However, the internal rate of return (IRR) also depends on the investment costs for the storage. The investment costs are depending on the capacity and the power [21].
For the model constructed throughout the next chapters I chose a Lithium-Ion battery (Li-ion), whose features are displayed in Table 2. LI-ions have high roundtrip efficiencies, low SDR and an acceptable life time respectively number of lifecycles [4], [18], [21]. Based on increased scientific and industrial activities, the price of Li-ion batteries is expected to drop substantially within the next years. Additionally, technological improvements due to the development of electric cars are

expected for this type of EES [24]. Their modularity allows fast deployment of functional EES units without special requirements on geology or infrastructure. Although PSPP are found to economically outperform Li-ion due to much lower costs [15], [21], their deployment is limited topographically and ecologically.[4] Given that the goal of this thesis is to evaluate the ability of MLAs to learn how to trade on the EM, the technology of the simulated EES is secondary.[7]

Table 2 Features of the chooses Lithium-ion EES based on the findings of *Thilo Bocklish* [25]; power and capacity are chosen independently.

Name	Value	Unit
Storage efficiency	0.825	DLU
Self-discharge-rate (SDR)	0.007	%/h
Power	250	kW
Capacity	1000	kWh

3 Evaluation Process

The goal of this thesis is the evaluation of ML-operated ESS trading on the German 'day-ahead' market with hourly resolution. To achieve this, the evaluation process is split into three parts. In the first part a linear optimization model (LPM) is formulated. It maximizes the revenue of an EES trading at the EM. Simplified, the operator can buy, store, and sell energy at different price levels. The result of the LPM is an array of optimal storage activities (OSA) (charging, discharging, waiting) for every hour of the years 2010 until incl. 2015. In the second part the array of OSA is used to label the market data for the same period. Subsequently, MLAs are used to train different classifiers on this matrix of labeled market data for the years 2010 - incl. 2014. The year 2015 is left out during the training process and is then used to determine the performance of the algorithm. This allows to simulate real market conditions where the agent has only the future information available that it would have under real market conditions.[8] The third part is the evaluation itself. A storage logic (SL) is used to simulate the performance of the classifiers acting as an agent controlling an EES under real market conditions of the year 2015. The separation of training/optimization of the MLA is necessary to avoid

7 In connection with the selection of the EES, a LP model was formulated to find the optimal EES and E2P ratios based on the data used by [25] for 7 different technologies. Although the EES traded optimally, no EES was profitable due to the high investment costs.

8 This includes any form of explicit or implicit information transfer from the 'unknown' data set. An explicit information transfer would e.g. be the information of the actual price at a future point of time. An implicit information transfer would e.g. be the calculation of the mean based on the complete data including the 'unknown' parts of the data.

overfitting and data leakage of the classifiers. Figure 3-1 shows the structure of the different data sets used during the process.

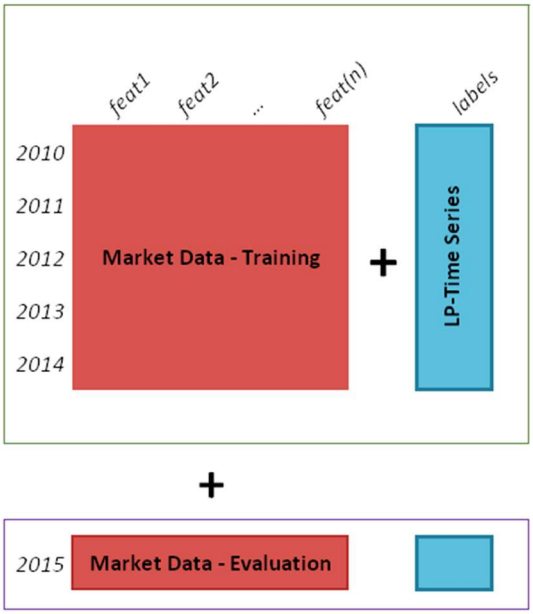


Figure 3-1 Structure of the datasets used during the evaluation process. The red frame represents the power market data, holding time series of the input features. The blue frame represents the results from the LPM providing OSA-labels to the corresponding values of a time step based on the EM data. The green frame represents the data used to train and optimize the MLAs.The purple frame represents the data used to evaluate the classifiers.

3.1 Data

The simulation of the EES under real market conditions requires real market data. In my thesis I use the data from the German EM, which is provided by the European Network of Transmission System Operators for Electricity (ENTSOE) [26]. The open source project 'Open Power System Data'[27] provides a respective API[9]. The data includes price and load levels as well as other additional timeseries (e.g. VRE generation). This is also the data used to train the MLAs. The data is formatted as a timeseries with an hourly resolution over the period of six years (2010-2015). The original dataset contains values for the columns listed in
Table 3 for every time step.
Table 3 also provides the most important statistical values for the market data. Both the wind and the solar generation show a large variance. Their median is significantly lower than the mean. This indicates outliers for higher generation levels. At the same time the minimum generation is (close) to zero.

9 Application programming interface.

Table 3 Original data for the German energy market OPSD [27] and the most important statsitical parameters (own calculations).

		Mean	Median	Variance	Min	Max
Load [MWh]	accumulated load for one hour	5 5423.76	55 278	1.04 * 108	29 201	79 884
Price [€/MWh]	price on the day-ahead market	38.49	38	2.64 * 10^2	-222	210
Solar generation [MWh]	accumulated hourly solar generation	2 599.95	0	2.52 * 107	0	26 055
Wind generation [MWh]	accumulated hourly wind generation	6 175.69	4 379	3.21 * 107	29	3 3626
Residual Load [MW]	remaining load after subtracting VRE generation	47 303.21	46 928	1.21 * 108	8 264	7 8070
Solar forecast [MWh]	accumulated hourly forecast for solar generation (all four TSOs)	3 708.77	167	3.24 * 107	0	2 6976
Wind forecast [MWh]	accumulated hourly forecast for wind generation (all four TSOs)	6 258.44	4 499	3.06 * 107	219	3 7322
Installed solar capacity [MWh]	installed solar capacity (all four TSOs)	2599.95	34 199	2.52 *107	10 47310	26 055
Installed wind capacity [MWh]	installed wind capacity (all four TSOs)	32 068.15	29 932	5.27 * 107	23 093	47 238

[10] Missing values neglected.

Figure 3-2 illustrates the installed capacities and generation levels for wind and solar power throughout the observation period. The installed capacities of both technologies grew constantly. The monthly average means illustrate the level of fluctuations of the VRE. Within the observation period the load varies between 79 884 and 29 201 MW with a mean of 55 423 MW. The residual load varies between 78 070 and 8 264 and has a mean of 45 460 MW. The maximal residual load is by a factor of ten greater than the minimal residual load. The flexibility of the power supply system must compensate these variations. The price is nearly normally distributed with a standard deviation of 16 (variance 263). The maximal price within the observation period is 210 EUR/MWh and the minimum price is -222 EUR /MWh.

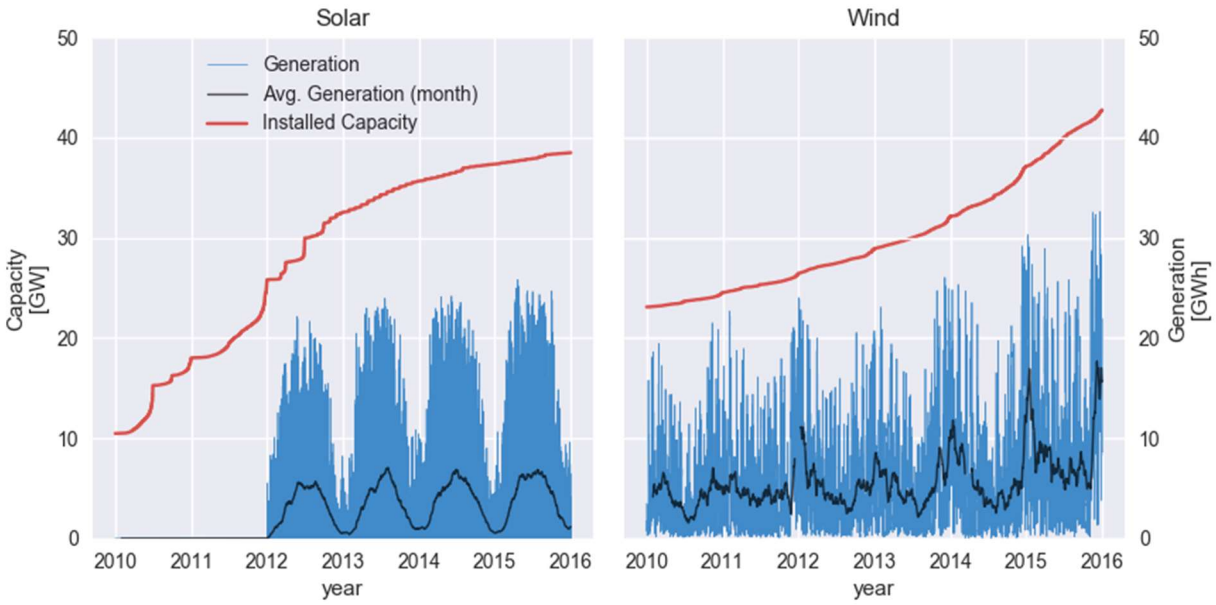


Figure 3-2 Installed capacities and genereration for wind and solar energy (The graphics share both y axes).

Within the model it is assumed that the EES acts as a 'pricetaker'[11]. Hence, the hourly price and the storage parameters (Table 2) are sufficient information to formulate the LPM and the SL model described in chapter 3.2.1 and 3.6 0without further manipulation. However, to facilitate ML processes, it is required to transform the information into a machine-readable format [28], [29].

11 The price taker assumption says that a single (small) actor on the market does not influence the market balance and therefore the price [15], [18], [49].

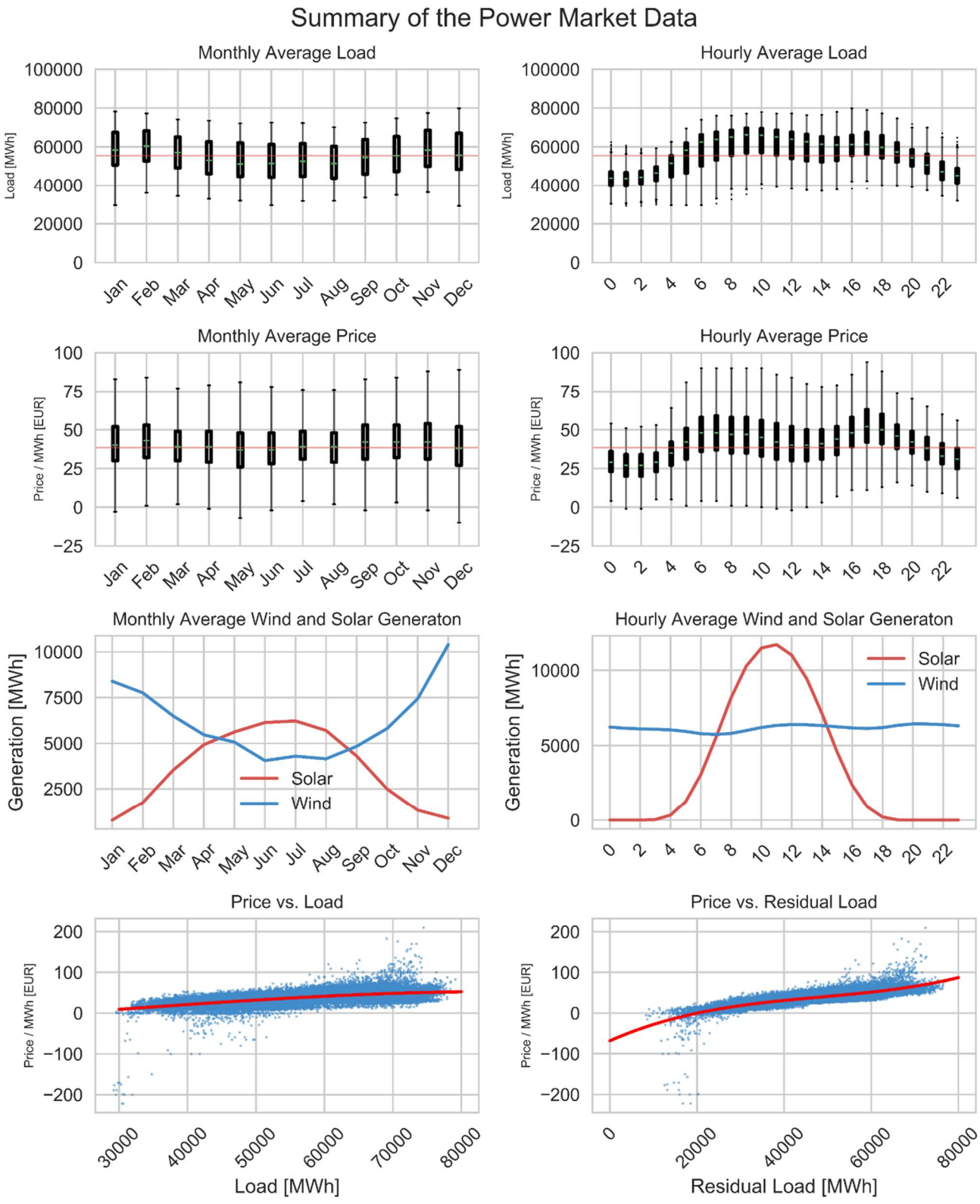


Figure 3-3 Analysis of the power market data (2010-2015); own calculations.

Figure 3-3 displays eight different graphical representations of the German EM data. The first row shows boxplots for the load, the second boxplots of the price. In the first column the boxplots are grouped by months, in the second by the hours of the day. During the winter months the average load is higher than during the summer months. The effect of increased demand during the winter months on the wholesale electricity price is moderate. However, during the winter months the variance of the price is clearly increased. The highest maximum prices were reached in February, the lowest minimum prices in December.
The hourly groups (second column) show two peaks during the day regarding the load. The first peak occurs during the morning hours (7 -10 am), the second during the afternoon respectively early evening (3-6 pm). Two corresponding price peaks reflect these load peaks. Compared to the seasonal fluctuations, the daily fluctuations are stronger. Under real-life market conditions this phenomena is acknowledged by the typical products traded on the EM [30].
The plots of the second row do not show the outliers in favor of a better graphical representation. There are some extreme positive and negative price spikes ranging from -222 to 210 EUR/MWh. Their prediction is very complex [11]. Nevertheless, the diurnal fluctuations of the load and price are a potential possibility for profitable arbitrage trading.
The third row shows the average wind and solar generation per month and per hour. The solar generation peaks during the summer months while the wind generation peaks during the winter months. During the day the solar generation naturally peaks during noon[12], while the wind generation is evenly distributed. These values are averages of volatile generation levels. Therefore, the observed levels for a single day can diverge substantially form the average values.
In the fourth row, the load (column 1) and the residual load (column 2) are plotted against the price. This results in a diagram which shows a relationship between price and quantity (price-quantity diagram). This phenomenon is amplified for the residual load. Simultaneously, the large variance suggests that there are additional factors influencing the price.

3.1.1 Additional, Derived Variables

Since the electricity price shows periodic behavior, it is important that the MLA interprets temporal information correctly. Originally, the timestamp was formatted as: 'YYYY-MM-DD HH' as a 'datetime-object'. This format was replaced by dummy variables for every year, month, and hour of the day. Additionally, a variable for the weekday was introduced, as electricity demand varies between weekdays and weekends.
Although some MLA can pick up very complex patterns it is helpful to 'highlight' relations within the dataset [28]. This is realized by adding new columns, holding metrics calculated based on information inherent to the data set. One of those metrics containing important relations is the 'residual load', which is defined as the load minus VRE generation. The residual load and the day-ahead price (price) have a positive Pearson correlation (0.87) compared to a correlation between

[12] Within the model I used the UTC time model. The peak of solar generation in Germany is therefore at 11:00 Greenwich meantime.

actual load and price of only 0.598. Further, the RM of the price, the load, and the residual load for 24 hours (RM24) and 168 hours[13] (RM168) where added to the data set as features. The RM takes the last 'x' values and calculates the mean of them, so x denotes the window size of the RM. Both RM values show a positive correlation to the price (RM24: 0.701, RM168: 0.56). Besides the correlation, the causal justification to add RMs is to account for short term price trends.

Table 4 Summary of all input variabes (features) for the ML process.

Name	Unit	Type	Specification
Load	MW	decimal	-
Price Day Ahead	€/MWh	decimal	-
Lagged Price Day Ahead	€/MWh	decimal	Lags: [-1…-24, -168]
Lagged Load	MW	decimal	Lags: [-1…-24, -168]
Rolling Mean Load	MW	decimal	Window size: [24h, 720h]
Rolling Mean Price	€/MWh	decimal	Window size: [24h, 720h]
Residual Load	MW	decimal	-
Solar Capacity	MW	decimal	-
Wind Capacity	MW	decimal	-
Solar generation	MWh	decimal	-
Wind generation	MWh	decimal	-
Renewable Generation	MWh	decimal	-
Total Forecast Solar	MWh	decimal	-
Forecast Wind	MWh	decimal	-
Forecast Solar	MWh	decimal	-
Forecast Total	MWh	decimal	-
Dummies for Hour		Boolean	[0-23]
Dummies for Month		Boolean	[1-12]
Dummies for Year		Boolean	[2011-2016]
Dummies for Weekday		Boolean	[1-7]
Workday		Boolean	-

3.2 Methods

3.2.1 The Benchmark Model as Linear Optimization Model

Linear optimization models are a popular method to study the different aspects of the energy markets [11] and are therefore a well-researched topic [12], [15], [18], [21], [31]. Linear optimization, respectively linear programming as a subdomain of operation research, describes the mathematical/analytical modeling of dimensioning-, logistic- and scheduling problems [32]. The underlying mathematical principle of LP is the simplex algorithm developed by *George Danzig* [33]. The simple algorithm finds either an optimal or unbounded respectively infeasible solution. LP models can describe storage scheduling problems. In this thesis the LP storage scheduling model is

[13] 24 hours times 7 weekdays

used to generate an optimal series of actions (charging, discharging, or waiting) of an EES and to create a series of OSA, which are then used as benchmark for the machine-learning algorithm.
In 2014, *Bradbury, Pratson*, and *Patiño-Echeverri* [21] published a paper in which they describe a LP model optimizing storage deployment at seven different U.S. energy markets. They analyze 14 different types of storage technologies to find the optimal P2E ratios. Their LP model maximizes the revenue of a merchant by arbitraging the spot market prices of 2008. As decision metric they used the internal rate of return (IRR). As a result the EES classify as profitable if the IRR is greater than 10% [21].
By calculating the overnight costs of the EES, consisting of the capital cost of power [$/kW], capital cost of capacity [$/kW], and the present values of the future revenues, *Bradbury, Pratson,* and *Patiño-Echeverri* can solve the IRR for different EES-types and P2E ratios. The findings of this study are that PSPP, compressed air energy storage (CAES) and high temperature batteries have the greatest potential for arbitrage trade and that the majority of EES will be optimally sized with an E2P ratio of 4 or less hours of energy storage [21]. Very similar model designs can be found in [12], [18].

3.2.1.1 Model Formulation

In this thesis I use a similar LPM as *Bradbury, Pratson*, and *Patiño-Echeverri* to optimize the storage revenue of an ESS under the day-ahead market conditions. The input data is a time series of hourly price values (π_t) in Euro per MWh for six years (see Chapter3.1).
The model simulates the storage activities of the agent maximizing the revenue by trading electricity at different price levels. The capacity of the agent is fixed at 1 MW to guarantee the price-taker assumption. The other technical parameters of the chosen battery are already described in Table 2. ρ describes the SDR due to parasitic losses. $P_{in}(t)$ describes the charged energy in [kW] while $P_{out}(t)$ describes the discharged energy [kW]. P_{in} and P_{out} are the controllable variables. Eq. 3-1 describes the objective function of the linear program, where π(t) describes the price [$/kW] at time t. η describes the roundtrip efficiency altering the costs of electricity to account for the losses during the storage process.

$$\max r = \sum_{t=1}^{t=876} \pi(t)\left[P_{out}(t) - \frac{p_{in}(t)}{\eta}\right]\Delta t$$

Eq. 3-1 Objective function maximizing the revenue (r) of an EES modified after [21].

The objective function is subject to the constrains:

$$E(t) = (1-\rho)\,E\,(t-\Delta t)[P_{in}(t) - P_{out}(t)]\Delta t$$

Eq. 3-2 Logic describing the (dis)charging process modified after [21].

$$0 \le P_{out}(t), P_{in}(t) \le P_{\max} \forall (t)$$

$$0 \le E(t) \le\ E_{max} \forall (t)$$

Eq. 3-3 Constrains limiting the maximal (dis)charging P and the capacity to the E2P corresponding ratio values.

The constrains are limiting the trading actives to the physical characteristics of the simulated ESS, where E_{max} is set to 1MW. P_{max} denotes the maximal energy flow during the (dis)charging process.

3.2.1.2 Model Realization

The program was realized in GAMS 24.8.4 and solved with the CPLEX algorithm (Appendix ii.d). The maximized revenue is used to evaluate and compare the performance of the MLAs. Additional information is stored in the time series of $P_{in}(t)$ and $P_{out}(t)$. These time series contain the values, used as labels during the training process of the MLAs.

During the modeling process several assumptions were made. The fundamental limitations of basic arbitrage analysis using LP and historical data is the assumption of perfect foresight, but there are also concerns regarding the technical robustness of the model due to the linearity of the problem [12], [18]. For example, the efficiency, SDR and the (dis)charge rates are assumed to be constant but under real world conditions, these values are not linear. Some ESS have nonlinear relations between power, efficiency and SOC or calendar age. Others (e.g. CEAS) show deployment delays [7]. Additionally, the maximal number of life cycles can be reduced by ignoring the recommended depth of discharge [4]. Combined, the assumptions in my model positively affect the ESS's economic performance. Thus, the results of the LPM act as an upper bounder for ESS arbitraging at the day-ahead market.

The intermediate results of the LPM are the foundation of the subsequent model. Solving the LPM results in a maximal profit an agent can earn by arbitraging on the German EM. This maximal profit is a benchmark respectively upper bounder for all following processes. Additionally, the LPM provides solutions (i.e. the optimal (dis)charging power) for the unknown variables $P_{in}(t)$ and $P_{out}(t)$. These time series are later merged into a single time series of optimal storage activities (OSA).

3.2.2 Machine Learning

ML is about extracting knowledge from data. There are several different algorithms available to fulfil this task, each one fitting best for specific questions and different dimensions of problems. Essentially, MLAs can be separated into three main groups: **Supervised** learning, **unsupervised** learning and **reinforcement** learning [28], [29], [34].

3.2.2.1 Supervised learning

Supervised learning is a kind of MLA that automates a decision-making process. A set of input and output pairs is provided to an MLA which finds a way to generalize from these known examples. Within the learning process, the MLA develops a function that can produce the output for a given input. This procedure does not require any additional human interaction. After a successful initialization, the MLA can produce an output for an unknown input. The input is called 'feature' and the output is called 'label' [29]. The names for the market data and the OSA were chosen accordingly. Supervised learning is the simplest and most comprehensive, hence the most promising form of machine learning, although the correct labelling of the input data is a crucial task during the supervised learning. If this is not possible, supervised learning is not an option. If applicable, it is the primary choice for ML applications [28].

3.2.2.2 Unsupervised learning

Contrary to supervised learning, unsupervised learning does not require labelled data. Unsupervised learning is a process used to detect unknown structures in a data set, e.g. by segmenting customers into different groups based on their purchases. Unsupervised learning is often used as a pre-process to supervised learning [28], [29].

3.2.2.3 Reinforcement learning

Reinforcement learning describes the process where an agent learns to choose correctly from a set of alternatives under different conditions based on a utility function. Decisions leading to a desired outcome (e.g. reduced costs) lead to an increase of the probability of choosing a certain alternative under a given situation [29].

Through the interaction with the environment, an agent can then use reinforcement learning to learn a series of actions that maximize its reward via an exploratory trial-and-error approach or deliberative planning. This form of learning is prominent for multi-agent based models and genetic algorithms [29].

For teaching an EES optimal storage-behavior supervised and reinforcement learning are potential approaches, since both can be designed to learn a profitable trading behavior. For reinforcement learning the reward could be trading profits. Supervised learning can be used to learn the optimal behavior by mimicking the results of an OSA.

3.2.2.4 Model Formulation

As mentioned above, supervised learning is the simplest form of machine learning, which is why it is used to formulate the following model[14]. Within the scope of supervised learning the market data represent the *features* and the LPM's outputs (OSA signal) represent the labels.

Depending on the question respectively the format of the label, the problem is solvable by either a regression or classification algorithm, so a distinction between 'classification' and 'regression' problems is made [28].A first examination of the OSA shows that most (85.2%) of the signals from the LPM are either 1 (charge with full power), 0 (wait) or -1 (discharge with full power). The remaining 14.8% are very close to these values, so a classification approach appears to be reasonable (see Figure 3-4). The small variance of the OSA is caused by the fact that the EES' capacity is a multiple of its power. Only the small impact of the SDR is disturbing a 'perfect' load cycle. However, the LPM uses the free capacities arising from imperfect load cycles to perform additional trades.

The OSA-signals for an EES with an E2P ration of 10/3 are illustrated in Figure 3-6. This ETP ratio leads to OSAs with reduced power. This effect would be further amplified if the charging and discharging power were not symmetric. For EES that show such kind of OSA patterns, additional

14 I chose to focus on supervised learning because it seemed more applicable. However, reinforcement learning also has the potential to deliver good results and should be investigated further in future publications [52].

(intermediate) classes would be required to adequately represent the optimal behavior[15]. However, in this case a distinction between the three classes (charge, discharge, wait) for an EES with an E2P of 4 is reasonable.

For the labeling-process, a simple algorithm (see below) was introduced. It classifies the OSA along a symmetric threshold into either charging (1), discharging (-1), or waiting (0). The optimal threshold was found by iterating over a list of 100 possible thresholds ($\mathbf{0}$, $\mathbf{0.01}$, $\mathbf{0.02}$, … $\mathbf{0.99}$, $\mathbf{1}$) and evaluating the resulting new OSA via the SL (see chapter 3.6). The optimal symmetric threshold was found to be within the bandwidth of equally optimal thresholds between 0.05 and 0.99. This large bandwidth also shows that there are no significant deviations from the classes 1, 0, and -1.

```
For each signal in OSA-signals:
       if signal >= threshold:
            class = 1
       if signal <= threshold * (-1):
            class = -1
       else:
            class = 0
```

Eq. 3-4 Pseudo code for the classification of the signals form the LPM.

15 Considering a PSPP with a natural inflow varying over time or an EES with larger SDR, the class boundaries become blurry. When investigating EES with such OSA patterns it might become reasonable to choose a regression approach.

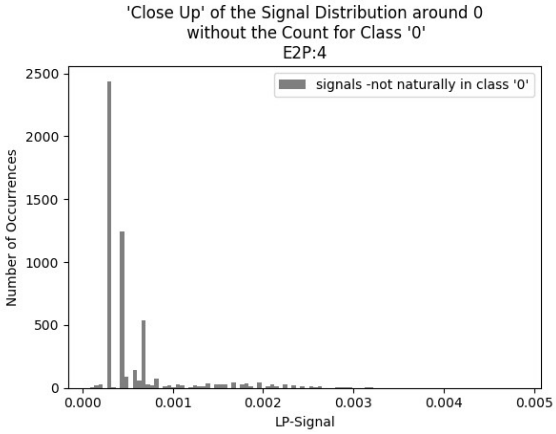


Figure 3-4 Illustrating the distribution of signals not 0 near 0. The x scale is limited to 0.005. Compared to the classes of -1,0, 1. Here naturally indicates the optimal signal from the LPM without any classification process.

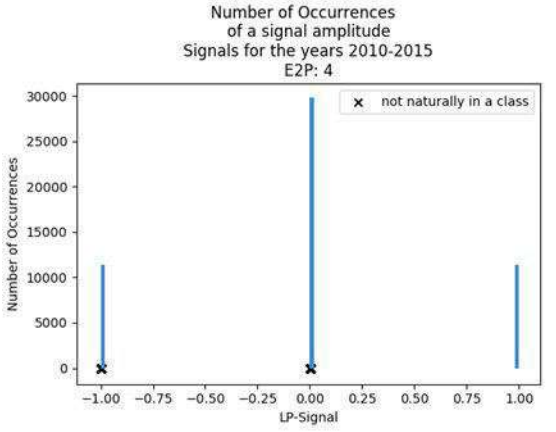


Figure 3-5 Number of occurrences of distinct OSA -signals by the LPM for an EES with an E2P ratio of 4. The black 'x' denotes values that are not exactly 1,0, -1. It is observable that there are three distinct groups of signals.

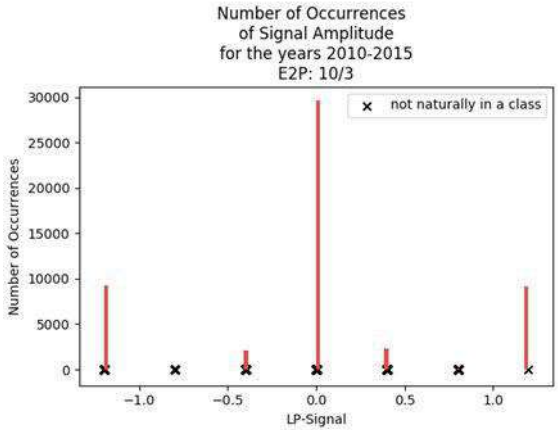


Figure 3-6 Number of occurrences of distinct OSA -signals for an EES with an E2P ratio of 10/3. The black 'x' denotes values that are not exactly part of one group. The number of groups is less obvious. Additional groups must be introduced to capture all signals correctly.

The classification includes a loss of information which inevitably reduces the performance of an agent trading based on this signal compared to the optimal solution of the LPM. A way to calculate this loss of performance is to use the SL (chapter0 3.6) simulating an EES trading based on the classified signal. The performance of the SL can then be compared to the LPM performance. This comparison allows to draw conclusions about the quality of the classification. Eq. 3-5 illustrates the calculation of the quality of the classification process.

$$relative\ performance\ of\ classification = \frac{classified\ OSA}{original\ OSA}$$

Eq. 3-5 Calculation of the relative performance of the classification process.

The optimal total profit calculated by the LPM was *52959.40 €.* After the transformation (Eq. 3-4) the SL achieves a total profit of *52863.76 €.* This is equivalent to 99.8% of the original profit calculated by the LPM.

The merging of the classified OSA-signals (COSAS) with the EM data set concludes the labeling process. The rows of the resulting table represent single hours for the time span of 2010-2015. The columns represent the features (load, price, wind, and solar generation, etc.), a single column with the COSAS is representing the label.

In this constellation, the label represents the COSAS for the timestep of the corresponding row. This approach does not reflect a realistic situation under real market conditions. For trading purposes, it is interesting to know which action should be taken next, according to the future market conditions. Therefore, the label is shifted one timestep into the past. This leads to a table where the features for t correspond with the COAS for t+1. In other words, the label is now the COAS for the following hour.

3.3 Preprocessing

The last step of the data manipulation and simultaneously the first step of the MLA optimization is the preprocessing. This describes the process where the data is manipulated to improve the algorithms' performance. The first step is the elimination of empty data fields. There are several options how to deal with missing data, e.g. deleting corresponding columns or rows, replacing it with a default value (e.g. *-999999*) or the mean of the column. Alternatively, the last valid value can be repeated (i.e. forward fill) [28]. To avoid data leakage, I chose to repeat the last value[16].

16 While filling empty cells with the mean value of a column, information of the whole column is necessary for this process. Filling with last valid values means that passed values are used to fill the cells. No information is transferred from the "future".

3.3.1 Train-Test Split

The following steps are performed with the machine learning library 'Sklearn' for python[17]. The split between the test and the training samples is important to avoid data leakage and overfitting. This means that any further optimization of the learning process is performed solely on a sub set (training set) of the parent population. This includes all typical preprocessing measures like 'scaling' and assures that no information of the unknown test data is transferred [28]. The data is split into a test set containing 20 % of the samples and a training set containing 80 % of the samples[28].
The training set is used to train the classifier and the test set is used to evaluate its performance. The split into training and test sample is determined by randomness, and thus not able to influence the learning process. However, it cannot be assumed that the split is unbiased, and that all classes/labels are distributed as they are in the parent population. The frequency of labels in the parent population is not evenly distributed. 56.5 % of the labels are '0' (wait) and each 21.7 % are 1 (charge) respectively -1 (discharge). To avoid biased training sets[18] , I assured that the labels are represented with the original frequency in both the test and training sets. This process is called 'stratification' [35]. The actual data split is performed during the cross-validation process. This is a form of train test split which minimizes the risk of biased training sets respectively hyperparameter tuning (see chapter 3.4.2).

3.3.2 Scaling

Many Algorithms are designed under the assumption that features have values close to 0 and are comparable in size. This accounts particularly for metric- and gradient based classifiers such as support vector machines and logistic regressions [28]. These classifiers expect standardized values (mean equals zero and variance equals 1). Unscaled data degrades the performance of an MLA by preventing or slowing down the convergence.
There are several methods of scaling data, differing by the approach on how to estimate the parameters used to shift the data [36]. To limit the scope of this thesis only two different scaling techniques with fundamentally different approaches are compared: the **standard scaler** and the **quantile transformer**. The scalers are part of the 'Sklearn preprocessing' module.

17 Sklearn is a high-level open source framework for machine learning and data preprocessing based on the NumPy library for vector and matrix calculations. Sklearn provides a large amount of different functions, which are highly adaptable for specific use cases [28], [29], [37].

18 A data set is 'biased' if a single label is overrepresented.

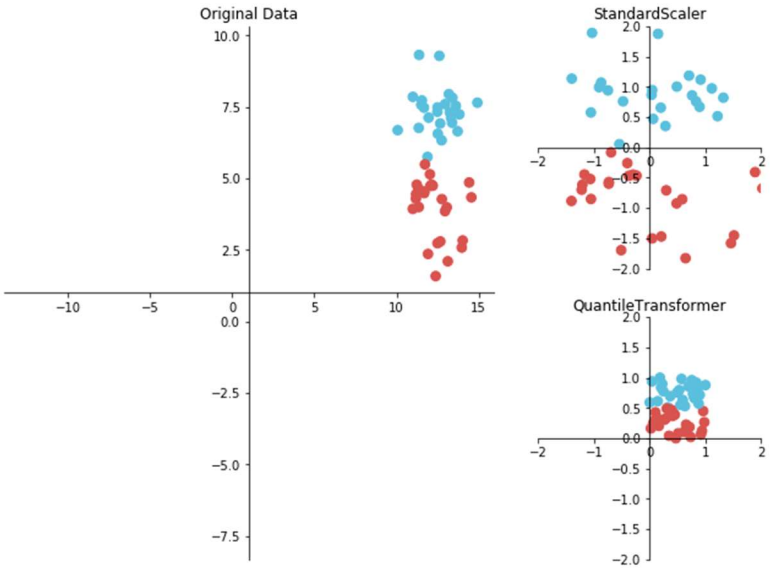


Figure 3-7 Schematic presentation of the effects of scaling on a two-dimensional dummy data set. Based on [28].

The standard scaler performs a linear transformation of the feature set by subtracting the mean and dividing the result by variance for every feature. The resulting feature has a mean of zero and a variance of 1. Because of outliers, the spread of the transformed data on each feature is different [37],[36].

$$X_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

Eq. 3-6 Standardization [29].

During quantile transformation a nonlinear function is applied to the features. After the transformation the probability distribution function of the features will be uniform within the range of 0 and 1. This spreads out the most frequent values of the feature and maps outliners to the boundaries (0,1). New data that falls below or above the fitted range will be mapped to the boundaries? of the distribution. As a nonlinear transformation it distorts all linear correlations between the features [37],[38].

3.3.3 Feature Interaction and Polynomials

Even after the scaling process, the performance of the classifier can still be enhanced. One method to further increase the performance is the introduction of artificial features. A common method to enrich the feature representations of the original data is the addition of interaction- and polynomial features. For a vector with two features (a, b) a second-degree polynomial expansion returns a vector with 6 features (1, a, b, a^2, ab, b^2). This method allows to represent interactions between features as a product and quadratic relations as square of a feature [28].

Nevertheless, the length of the resulting vector increases depending on the length of the input vector (n) and the degree of polynomial expansion. After the expansion the resulting vector has the length $\binom{n}{d} * 2 * n$. This is not trivial. For the input data set with the shape (52824, 119), the second-

degree polynomial expansion results in a feature vector with a length of 7 260. Some algorithms (e.g. K-nearest neighbor) are not designed for such high dimensional data.
Many of the original features are dummy variables representing the temporal dimensions (1 if Monday, 0 if not Monday). The combination of two dummy variables describing the same feature (e.g. Monday and Thursday) result in columns of constant zeros. These columns contain no additional information and can be removed without hesitation.

3.3.4 Feature Selection and Extraction

Complex models (i.e. containing many features) tend to overfit compared to more general models [28]. Hence complexity/dimension reduction is a possibility to improve the model. After the feature expansion, the data set is rather sparse and probably partially redundant[19]. This degrades both the computational[20] as well as the classification respectively generalization performance [34]. The dataset also includes the products of the dummy variables resulting in columns of constant zeros which must be removed. Moreover, other features (columns) of the data set could also contain no additional information-explaining label. Thus, these feature dimensions do not improve the classification quality of the model. During the training process indeed even random data (e.g. no relation to the label) can be used to increase the training accuracy, but predictions based on these features will certainly decrease the models' prediction quality (Overfitting). These additional dimensions also reduce the computational performance during the training process. This accounts especially for larger data sets (see chapter 3.3.3). Firstly, the initial selection is based on 'arbitrary' causal connections. Usually a closer examination of the feature dimensions' suitability helps to improve the model's quality. Secondly, the polynomial enriched data set exceeds the mathematical respectively computational capabilities of some MLA [29].
I therefore used a set of different techniques to reduce the number of features (i.e. the number of dimensions of the data). Four different approaches are examined in greater detail: the reduction of features based on univariate statistical tests (e.g. ANOVA), the so-called 'model-based approach' and the principal component analysis (PCA). The fourth approach is the regulation-based feature selection, which can be applied within the logistic regression and is discussed in chapter 4.5

3.3.4.1 Univariate Statistical Tests

The first option is the reduction of features based on univariate statistical tests (e.g. ANOVA). The statistical significance between feature and label is computed and an arbitrary number of features with the highest confidence is selected. This arbitrary number is often based on the p-values of the features. This assures that only features with statistical significant influence on the label are considered by the model and 'useless' features do not degrade the performance. This approach tests features only one by one. Possible interactions with other features remain undiscovered [28]. Because of this neglecting of the interactions between features, this approach is not further pursued.

19 The data matrix contains many zeros.

20 Some MLAs are exceeding the capacities of 16 GB RAM while training on this dataset.

3.3.4.2 Model-Based Approach

The second approach is a so-called 'model-based approach'. Some models (e.g. Random Forest, Logistic Regression) can be used to estimate the importance of single features. One of them is the 'random forest' model (see chapter 4.3 for further details). While training the random forest model, the importance of features is determined based on the decisions of the trees. This so called 'Gini-importance' shows how much impurity/entropy reduction can be obtained by a (data) split based on this exact feature [39]. Still, because decision trees are determined by randomness, this approach is neglected.

Although this approach was neglected, the model-based feature selection provided some insight into the importance of individual features as well as feature combinations. The results are therefore elaborated in Appendix i.b

3.3.4.3 Principal Component Analysis

While the univariate statistical tests and the model-based approach where supervised heuristics to select features based on certain criterions, the last method *extracts* features. The principal components analysis (PCA) is a popular approach for deriving a low-dimensional set of features [34]. It identifies patterns within data based on the correlation between features trough finding the directions of maximum variance in high-dimensional data. These features are projected on hyperplanes to reduce the number of dimensions while still describing most of the variance [29], [39]. However, during the projection information is lost. There is a trade-off between the number of features and explained variance, which *James* et al. [34] (2007) describe as following:

> *'Unfortunately, there is no well-accepted objective way to decide how many principal components are enough. In fact, the question of how many principal components are enough is inherently ill-defined, and will depend on the specific area of application and the specific data set. On the other hand, if we compute principal components for use in a supervised analysis[…], there is a simple and objective way to determine how many principal components to use: we can treat the number of principal component score vectors to be used in the regression as a tuning parameter to be selected via cross-validation or a related approach.'*

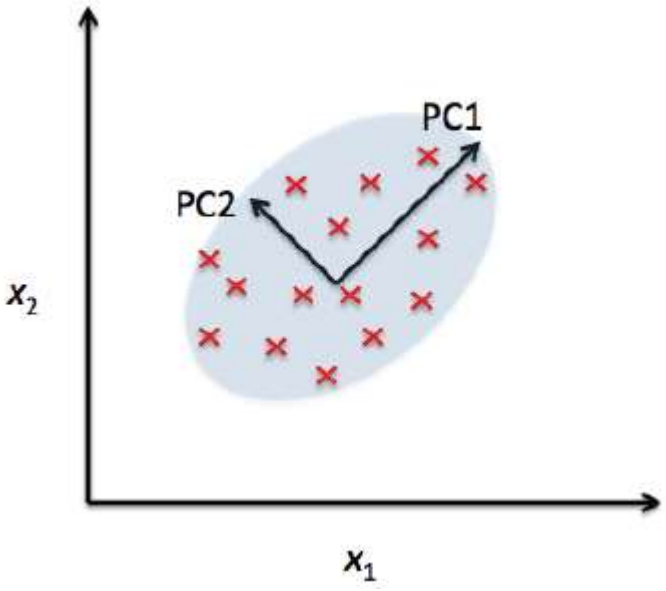


Figure 3-8 Principal components of x1 and x2, Graphic by [29].

Figure 3-8 displays the principal components for a two-dimensional dataset. The principal component 1 (PC1) is the line along the greatest variance of the features *x1* and *x2*. The principal

component 2 (PC2) is the orthogonal to PC1. PC1 alone can be used to describe most of the variance between *x1* and *x2,* while simultaneously reducing the dimensionality by 1. PCA does consider the labels of the data points, therefore the transformation does not improve the quality of the data imperatively. The measure for the PCA's quality is the explained variance [29].
The results of applying PCA on the training data are illustrated in Figure 3-9. The explained variance (EV) per feature is a measure for the additional information by each component. To display the results for the original features and the polynomial together, the relative values are displayed. The middle graph illustrates the cumulative EV.
For the original market data 84 of 119 principal components are necessary to explain 99.9% of the variance. For the polynomial data, 1208 of 7260 principal components are necessary to explain 99.9% of the variance. These sizes allow conscientious training of the classifiers. The bottom graph illustrates the absolute explained variance. It can be seen that the variance within the polynomial data set is clearly greater.

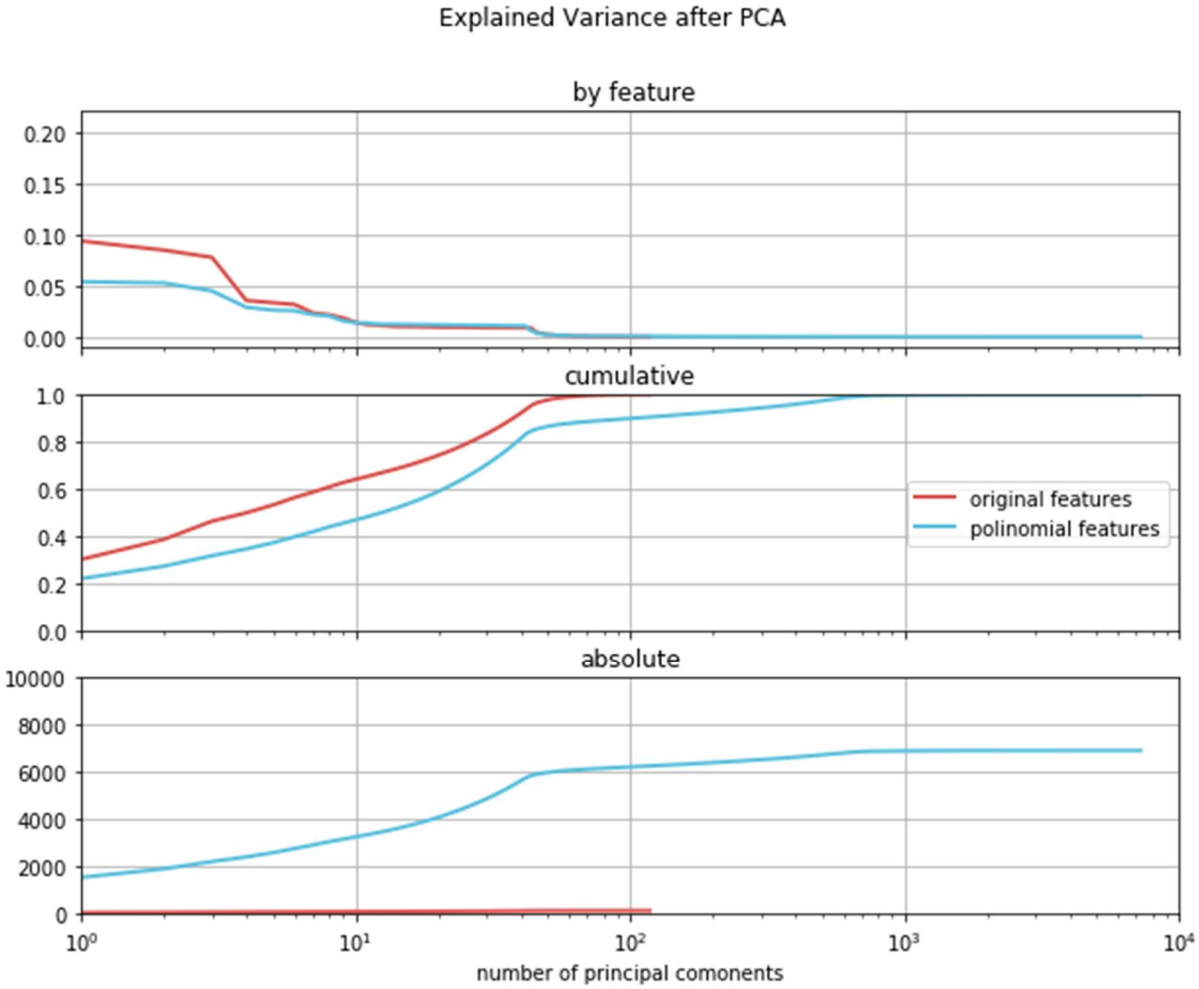


Figure 3-9 Top: Relative explained variance by principal component in descending order; Middle: Cumulative relative explained variance; Bottom: Absolute cumulative variance.

The goal of the feature enrichment was to add variance by introducing new combinations, potentially explaining the label. The feature selection/feature extraction assures the quality of the feature set and concludes the data preprocessing. The next step is the training, testing and optimization of the classifiers.

3.4 Optimization Framework

MLAs have parameters which are independent from the data input (penalty factors, choice of solvers, tolerance levels, etc.). These parameters are called 'hyperparameters' and allow the adaption of the algorithm to specific problems [29]. Since the goal of ML is to generalize from known data to unknown data, the performance of a classifier is defined by its potential to predict unknown data correctly (see chapter 3.2.2).

To test this potential, the labeled data is split into two sets - one for training and one for testing. After a classifier is trained on the training set it is tested against the test set. The quality of its predictions (e.g. classification accuracy) is calculated based on the results of the test set. If the quality meets the requirements, the process ends here.

It is very unlikely that the first attempt will deliver the best results for a problem [28]. To improve the quality of a classifier, the stepwise adaption of the hyperparameters is required. Setting the hyperparameters correctly is a core task within any ML-process [28], [29]. Depending on the number of adjustable parameters and conversion time of the MLA, there are two meta heuristics to find the (semi)optimal setting for the hyperparameter, called 'hyperparameter tuning': **grid search** and **random search** [28], [29].

3.4.1 Hyperparameter Tuning Heuristics

The grid search uses lists of options for hyperparameters. All possible combinations within these lists are tested. This assures that the best combination of the provided options is found. If there is a better setting of hyperparameters but it's combination is not in the provided lists of options (parameters), it will not be discovered. This process is time consuming, especially if there are many combinations of hyperparameters [40].

To reduce the computational time, a random search can be used instead of a grid search. A random search also uses lists of options for the specific hyperparameters, but instead of testing all possible combinations, only 'x' random combinations are tested, where 'x' represents a chosen budget of trials. Random search is used to determine which settings are promising for further investigation [40].

3.4.2 Cross validation

As already mentioned in chapter 3.2.2, the quality of a classifier is determined by its potential to classify 'unseen' data correctly. Any information transfer not available to the classifier under real-world conditions between training and test data set must be prevented, otherwise the quality of the classifier will be overestimated. This includes the scaling process, the feature selection, and especially the hyperparameter tuning.

If the hyperparameters are optimized against a single test, the results are likely to be adapted only to the characteristic of this specific test set. Therefore, the model loses its ability to generalize and a biased set of hyperparameter is selected. A common technique to avoid this biased hyperparameter selection is the so called 'k-fold cross validation', also simply referred to as 'cross validation' [28].

The 'k-fold cross validation' describes a process where the data is split into k same-sized subsets from which one set is selected to be the test set. The other sets get combined into a training set. The classifier is then trained and tested based on those two sets. In the next step the second set is selected to be the test set and so on. This process is repeated until all sets have been used as a test set (i.e. after *k* rounds). The quality of the classifier is the average performance over these *k* training and testing cycles. In combination with grid search (see chapter 3.4.1**Fehler! Verweisquelle konnte nicht gefunden werden.**), the set of hyperparameters with the highest average score is selected [29].

The advantage of cross validation is the reduced risk of hyperparameter selection based on random events. It is possible that a randomly selected test set is very similar/different to the training set if a set of hyperparameters is chosen or dismissed because of a single test result in a 'miss assessment' [28], [29]. The cross validation also increases the computational effort by the factor of *k*, which is only partially parallelizable [41].

Figure 3-10 illustrates the schematic procedure of the cross validation and the calculation of the resulting mean accuracy:

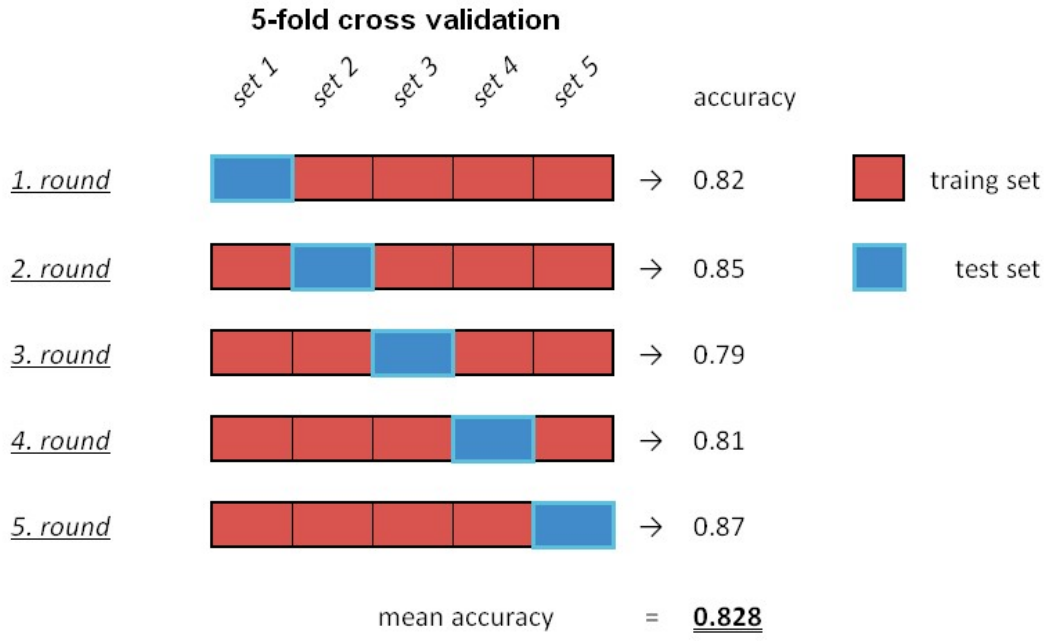


Figure 3-10 Illustration of an k-fold cross validation for k= 5. Every round includes a training a testing process.

3.5 Model Evaluation

The conclusive part of the evaluation process is the evaluation of the MLA's performance. The general process of the model evaluation is illustrated in Figure 3-11. The green frame represents the parts of the process optimized by grid respectively random search including the cross validation.

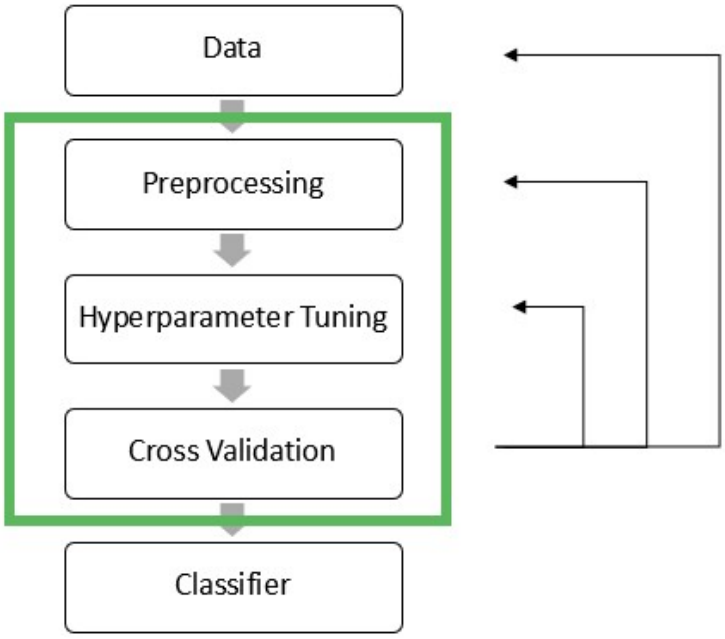


Figure 3-11 Procedure of the Classifier optimization process.

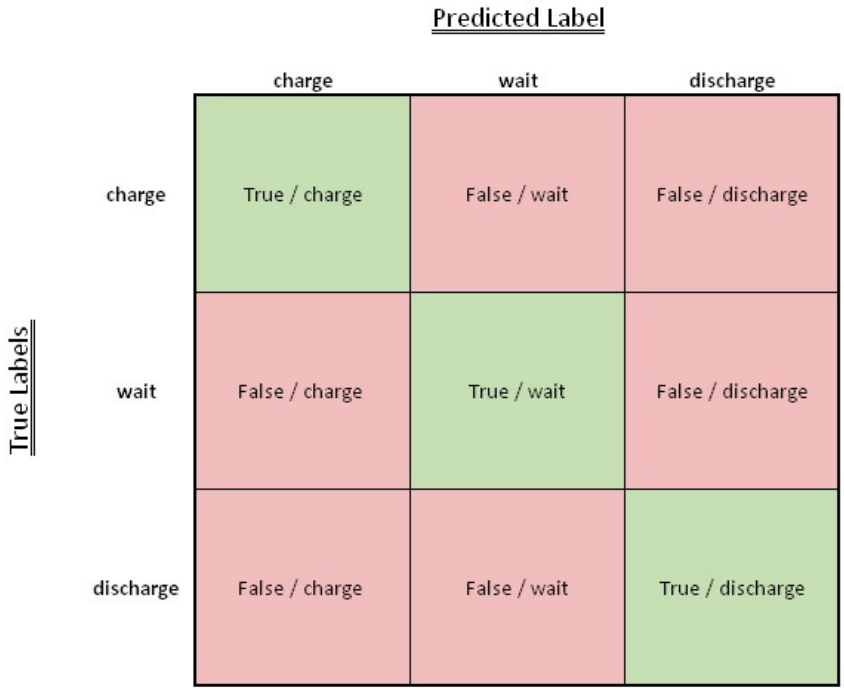
Given an MLA with two hyperparameters with each five options and two different scaling options, three different methods of feature selection are possible. This totals in *5 * 5 *2 *3 = 150* different sets of parameters. Additionally, performing a 5-fold cross validation results in 750 models trained and tested. This is only acceptable if the convergence time of a single model is low (e.g. several seconds). This is aggravated by the fact that most classifiers have more than two parameters including continuous variables like penalty values [28]. If the optimized classifier does not fulfill the classification standards, a new attempt with a new set of hyperparameters, other preprocessing techniques or new / additional data is necessary [29]. If the classifier fulfills the requirements, the process is finished.

3.5.1 Scoring Values

As mentioned above (see chapter 3.2.2), the test score of the classifier determines its quality. The default score when evaluating a (multiclass) classifier is the accuracy. The accuracy represents the percentage of correct predictions [29].

The accuracy is a decent and naturally comprehensible metric of evaluation, but since the classes in the original data (and probably in new data too) are unevenly distributed, the classifier tends to over-represent the most common class (55% wait, 22,5% each charge and discharge). A classifier

predicting '*wait*' in 100% of the cases has an accuracy of 55% without providing any additional information or benefit.



To counteract this behavior, two additional metrics are introduced: the precision and the recall. These metrics can be derived from the confusion matrix (Figure 3-12**Fehler! Verweisquelle konnte nicht gefunden werden.**) of true values and predicted values. The main axis of this matrix displays the correct classifications (i.e. the prediction is correct), all other combinations represent misclassifications.

Figure 3-12 Confusion matrix for the predicted and true OSA.

These metrics are derived from binary classification problems, so the standard formulas for those metrics must be altered by a weight factor. The first metric is the *precision*. The precision of the prediction of one class is the quotient of the number of correct predictions for one class and the sum prediction for that class (in 'Trues' of column / sum of column) [29]. For example:

$$Precision(charge) = \frac{(True/charge)}{(True/charge) + (False/charge)}$$

Eq. 3-7 Calculation of the precision [29].

The total precision is the weighted mean of the three individual precisions[21]. The weight is based on the number of true values for this class. The second metric is the recall. It describes the quotient of

21 There are different ways to compute the total precision/recall. The weighted mean is typically used for imbalanced classes [28], [29], [53].

correctly predicted true values for a class and the total number of true values for this class (Trues of row / sum of row) [29]. For example:

$$Recall\ (charge) = \frac{True/charge}{(True/charge) + (False/wait) + (False/discharge)}$$

Eq. 3-8 Calculation of the recall [29].

The total recall is the weighted mean of the three individual recalls. Independently, both metrics can be optimized to be 1 (i.e. perfect). The f1 score, the weighted harmonic mean of precision and recall combines those two measures [42]. Therefore, the f1 score is the standard additional representation of the classifiers quality:

$$f1\ score = \frac{2 * precison * recall}{precision + recall}$$

Eq. 3-9 Calculation of the f1-score [43].

These scores are helpful to statistically describe the quality of the classification. Still, the correct classification is only a simplification of the true objective: the maximization of the profit. Although the confusion matrix and the score metrics provide insight in the kind of errors made, they do not quantize the effect of a mis-classification monetarily.
Based on this metric, errors with little effect on the overall economic performance of the EES (e.g. charging at the second-best point of time instead the best) are penalized by the algorithm equally as major errors (e.g. discharging if charging would be the 'correct' option). This shows to be a fundamental problem during the optimization task.
The model was optimized by the means of an error score, but the actual objective function is the maximization of the profit. To take this into account, I assume a correlation between correct classification, quality, and profit. The examination of this correlation is part of the results which are discussed in chapter 5.3.
As shown, there are many options and possibilities to improve (or worsen) the quality of the classification within the entire process of developing a good classifier, many of them influencing each other. As the investigation of all those possibilities is not within the scope of this thesis, I applied a para-systematically approach by simply testing different settings and observing the effects. The second step was then the investigation of the interactions of single parameter combinations which resulted in two dimensional matrixes of results. Eventually this process of trial and error led to a set of preprocessing actions and hyperparameters small enough to be systematically examined via grid search as shown in chapter 5.

3.6 Evaluation Framework

The accuracy of the classification does not significantly determine the performance of an agent trading on the EM based on the classifier's predictions. An incorrect classification does not determine how economically wrong the decision is. To test the classifier's performance on the real market, I introduced a framework, written in Python (Appendix Appendix ii.c). Within this framework, the classifiers are used to act as an agent controlling an EES, which places the results of the classification into an economic framework.

Based on the market data of the year 2015, the classifiers predict storage behavior respectively storage signals. These signals are then used to trigger actions of an agent trading on a simulated EM. If the classifier predicts 'charge', the agent tries to buy electricity and charge the battery if there is capacity left. If the classifier predicts 'discharge', the agent sells electricity. If the classifier predicts 'wait', the agent takes no action. The following pseudocode describes to process in detail:

```
For (timestep) in all evaluation period:
       current market conditions = market conditions (t)
       signal = Classifier.predict_behavior(Current _market conditions)
       Current price = price (t+1)
       if signal is charge:
            free capacity = maximum capacity – current storage level
            buyable electricity = min(maximal charging speed , free capacity)
            current storage level = current storage level + buyable electricity
            balance = balance – buyable electricity / (storing efficiency) *
            current price

       if signal is discharge:
            sellable electricity = min (maximal discharging speed, current
            storage level)
            current storage level = current storage level - sellable electricity
            balance = balance + sellable electricity * current price

       if signal is wait:
            do nothing

       current storage level = (1-self discharge rate) * current storage level
```

Eq. 3-10 Pseduo code describing the storage logic (SL).

This process is repeated for every hour of the evaluation period. The market conditions of the current hour are used as an input for the classifier. The classifier predicts the action for the subsequent hour (t+1), the current price is set to the price of the subsequent hour.

If the predicted action is 'charge' (buy), the agent calculates the maximal amount of free capacity of the EES. He then buys this maximal possible amount. The maximal possible amount is limited by either the maximal charge speed or the remaining free capacity. The bought electricity is then added to the EES's current storage level. The last step is the update of the EES's balance. Analog to the calculation of the LPM (chapter 3.2.1), the costs of the bought electricity are the amount of electricity divided by the efficiency times the current price. Those costs are subtracted from the total balance.

If the predicted action is 'discharge' (sell), the agent calculates the maximal sellable amount of electricity, which is either limited by the maximal discharge speed or the remaining stored energy. The storage level gets updated accordingly. The last action is to update the balance. The profit from selling energy is the maximal sellable amount times the current price.
If the predicted action is 'wait', the agent takes no action.
The last action of a timestep is always the calculation and deduction of the self-discharge according to the self-discharge rate. The performance of the agent is compared to the performance of an agent using the OSA-signal from the LP model.

3.7 Other Strategies

As mentioned before, using EES for arbitrage trading is a popular strategy within the scientific community [15]. But since the simulation of EES's with LP requires perfect foresight, other strategies for arbitrage trading have emerged, many of which are used to simulate real world storage strategies. This helps to compare the results of the classifiers and to put them into perspective. In a 2016 paper *Zafirakis* et al. [15] examine the value of arbitrage trading of energy storage and introduce inter alia methods for the storage control.
Two of those methods are used in this model to simulate state of the art storage strategies: the weekly and daily 'back to back' strategy. The back to back strategy suggests using the same charge and discharge pattern as it would be optimal for the previous time period. This assures that the daily, weekly, and seasonal patterns of the electricity price are represented accordingly by shifting the OSA by 24h (daily) and 168h (weekly). The resulting series can be treated like the prediction signals of any other classifier.

$$shift\ (week)_t = OSA_{t-16}$$
$$shift\ (day)_t = OSA_{t-24}$$

Eq. 3-11 Calculation of the back-to-back strategies.

4 Classification Algorithms

4.1 Introduction

The classification algorithm is the centerpiece of every machine learning process. Within the Sklearn library the algorithms are organized in python classes [37]. The architecture of these classes is uniform as inheritance is used, which makes it easy to switch between different algorithms. The classification algorithms themselves have distinctive inner workings and requirements on the data preparation. This especially accounts for scaling and data size. The following chapters introduces six different kinds of classification algorithms and their implications for the problem tackled in this thesis.

4.2 K-nearest Neighbors

The K-nearest Neighbor (KNN) algorithm is a classification algorithm. Strictly speaking, the KNN does not really 'learn' and therefore is often referred to as 'lazy learner' [28]. During the training process the classifier stores all feature-label pairs of the training data set. During the prediction the distance between the feature vector and all known samples is calculated. The k training samples with the shortest distance to the new sample are chosen to determine the class of the unknown data. The most frequent class within these k nearest samples is the predicted class. Parameter k, the number of neighbors, is the main hyperparameter of the KNN algorithm. It determines the number of samples used to predict a new data point and is the key parameter to reduce the effect of overfitting. A small k leads to complex models with volatile decision boundaries that tend to overfit on the training data. A classification based on too many neighbors reduces the ability to correctly classify the data, because the distance to the farthermost considered neighbors gets too large [28].
Two additional parameters, the calculation metric, and the weight of the distance, can be used to optimize the KNN-algorithm. The calculation metric for the distance influences the selection of the neighbors. There are two basic options for calculating the distance: the **Manhattan** distance and the **Euclidian** distance.
The Manhattan distance is the sum of the absolute distances in every dimension:

$$d = \sum |x - y|$$

Eq. 4-1 Calculation of the Manhattan distance [44].

The Euclidian distance is the length of the straight line between the two points:

$$d = \sqrt{\sum (x - y)^2}$$

Eq. 4-2 Claculation of the euclidian distance [44].

The second parameter (weight of distance) is the weight of the votes of each nearest neighbor. By default, the weight of the votes is uniform (i.e. independent from the distance to the new data). Alternatively, the weight of the neighbor can be set inverse to its distance to the new data point. This reduces the influence of distant points but is still within the closest *k* neighbors (distance) [44].
An effect often observed in context of KNN is the 'curse of dimensionality', which describes the phenomena of feature spaces becoming increasingly sparse for an increased number of dimensions and a fixed number of observations [29]. Also, the KNN does not scale very well. The computational effort (O) of the standard (brute force) KNN classifier grows in O [DN^2], where N are the samples and D are the features [45]. The KNN is therefore not suited for a large set of input data with polynomial features. However, by limiting the training data to a fixed number of observations (i.e. *8 760* for one year of training data) and dispensing the option for polynomial features, the computational effort is manageable.

Figure 4-1 displays the effect the number of neighbors has on the classification quality. The blue line describes the score based on the training set, the red line describes the score based on the test set. A visual inspection of the plot shows the maximal score for the test set lies within the range of ~5 to ~30 neighbors.

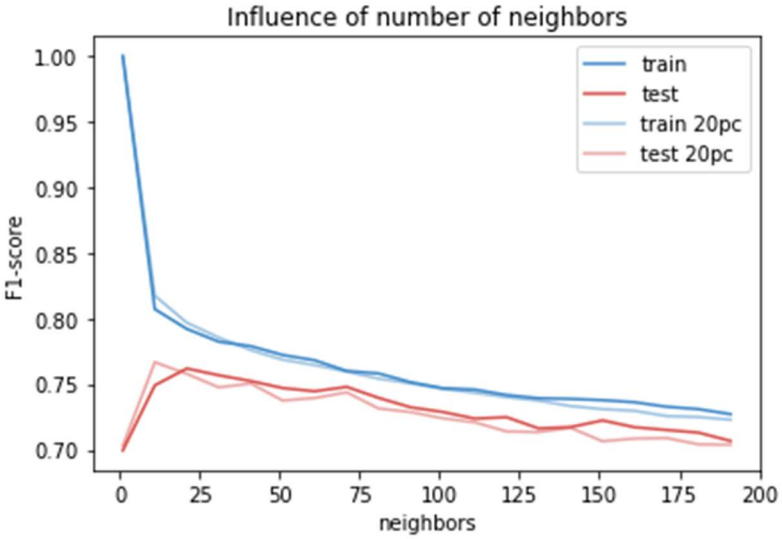


Figure 4-1 Influence of number of neghbors for an KNN classifier on the accuracy of the classification. 20pc denotes a feature set determined by

For a decreasing number of neighbors, the test score drops, and the training score raises rapidly. This is a sign of overfitting (i.e. a model with high complexity) [28]. With increasing numbers of neighbors, the classification quality slightly decreases.

Additionally, the same number of neighbors were tested after a PCA extracted 20 principal components (see chapter 3.3.4.3). The negative effect of the dimension reduction is small (compare Figure 4-1 pale lines). Based on these findings a grid search for a narrower hyperparameter space was performed to find the optimal hyper-parameter composition. Table 5 lists the hyperparameters and preprocessing steps used during the optimization process of the KNN-classifier.

Table 5 The parameter grid used to optimize the KNN classifier.

Parameter	Values
Polynomial Features	`[No]`
Scaler	`[Standard, Quantile Transformer]`
PCA (components)	`[20,40,50,100,119]`
Number of neighbors (k)	`[5, 10, 11, 13, 16, 20, 25, 30]`
Distance metric (p)	`['Manhattan', 'Euclidian']`
Weights	`['uniform', 'distance']`

4.3 Decision Trees

Decision Trees are a series of 'if-else' decisions ultimately leading to a classification. Every question splits the samples into one node ('parent') and two subgroups ('children') with reduced impurity. This split is orthogonal to one feature axis. This process is repeated until all samples in the last/terminal nodes are from a single class. In other words, the terminal nodes are 'pure'. Repeating the same series of questions for a new data point leads to a predicted classification [28], [39].
The underlying objective function aims to maximize the information gain (IG) at each (binary) split where D_p is the parent node, D_{left} and D_{right} are the children:

$$IG(D_p, a) = I(D_p) - \frac{N_{left}}{N_p} * I\left(D_{left}\right) - \frac{N_{righ}}{N_p} * I(D_{right})$$

Eq. 4-3 calculation of the information gain [29].

N represents the number of samples in the respective node, I stands for an impurity function. The information gain is the difference between the impurity of the parent and the sum of the impurities of the children. The impurity or splitting criteria is a metric of the difference within the sample classes [29].

Two different metrics can be compared: The **entropy** and the **Gini-coefficient**. The entropy is a metric measuring the impurity of a group. The node's entropy is 1 if the distribution is uniform. The entropy for a pure node is 0. The entropy criterion must therefore be minimized while building a tree. The entropy (IH) for the node (t) is calculated in Eq. 4-4, where i is the number of samples belonging to a class and c is the total number of classes. $p(i|t)$ is the relative proportion of samples of a class in the node t.

$$I_H(t) = -\sum_{i=1}^{c} p(i|t) \log 2(p(i|t)$$

Eq. 4-4 calculation of the entropy [29].

The second metric is the Gini-coefficient. This coefficient is a measure of impurity, which is why it must be minimized like the entropy during the trees construction. The Gini coefficient (I_G) is calculated in Eq. 4-5.

$$I_G = 1 - \sum_{i=1}^{c} p(i|t)^2$$

Eq. 4-5 calculation of the Gini coefficient [29].

The decision-tree-algorithm, in its basic version, continues until all terminal nodes are pure and every sample of the training set is classified correctly [28]. This is a prime example of an overfitted classifier. To reduce this effect, the construction of the tree is usually constrained [39]. There are

multiple ways of constraining the growth of a tree. Two of them – the **maximal depth parameter** and the **minimal number of samples** in a terminal node – are examined in greater detail.
The maximal depth parameter limits the size of the tree by constraining the length of the series of splits. The minimal number of samples in a terminal node reduces the effect of outliers. Based on their algorithm, decision trees do not require scaled input data [28]. Table 6 lists the hyperparameters and preprocessing steps used during the optimization process of the decision tree.

Table 6 The parameter grid used to optimize the decision tree.

Parameter	Values
Polynomial Features	[No]
PCA (components)	[No]
Maximal depth	[100,50,25,10]
Minimal samples for an additional split	[2,50,100,200]
Minimal samples in terminal node	[1,20,40,60,100]
Impurity metric	['Gini', 'Entropy']

4.4 Random Forest (RF)

Another method to improve the prediction quality of a decision tree is to train multiple decision trees on a slightly different data set (e.g. a bootstrapped data set). The predicted class is the result of voting by each single tree. This technique is called 'random forest'. A random forest is more robust against outliers and increases the generalization quality of a classifier [28], [29].
There are three additional hyperparameters examined for random forest. One is the **number of trees deployed**, the other two determine the **randomness of the single trees**. At every split only a randomly chosen fraction of features is selected (maximal features). A low number of features leads to decreased impurity reduction, simultaneously increasing the randomness, and consequently reducing overfitting. The last hyperparameter is impurity metric. The single trees of a random forest do not have to be limited in their growth, since the forest itself is quite robust against the noise of the single trees [29]. Table 7 lists the hyperparameters used during the optimization process of the RF.

Table 7 The parameter grid used to optimize the random forest.

Parameter	Values
Impurity metric	['Gini', 'Entropy']
Number of trees	[10, 20 ,50, 100, 500, 1000]
Maximal Features	[10,30,60,90]

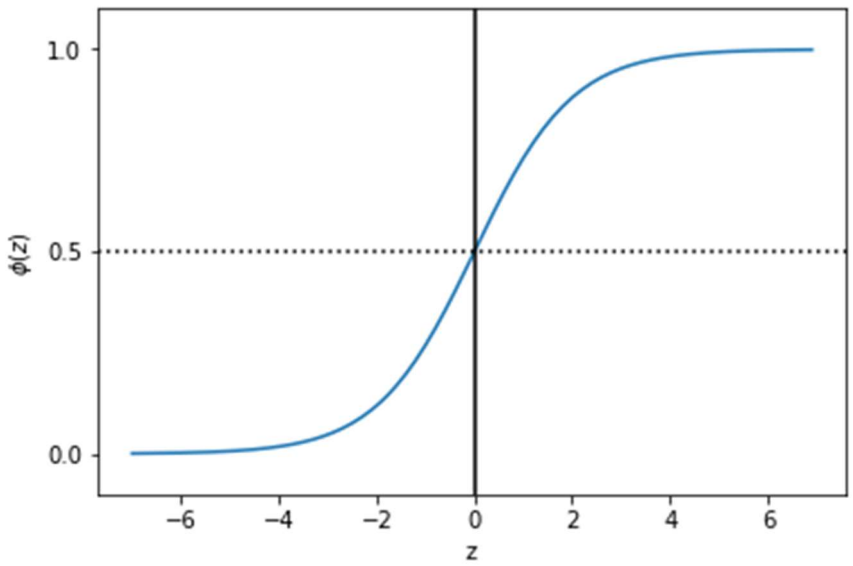


Figure 4-2 Sigmoid function for z[-7,7], [25].

4.5 Logistic Regression

The logistic regression (LR) is a binary classification model for linear separable classes. The formula used to make predictions resembles the formula for the common linear regression (see chapter 3.2.1). While linear regression estimates a value for a given set of features, logistic regression estimates the odd ratio of a set of features being part of a certain class $[\frac{p}{1-p}]$. The logistic function (logit) is the logarithm of the odds function. The logit function takes input values within the range of 0 to 1 and transforms them into values over the entire space of real numbers, which can then be used to express a linear relationship between feature values and the log-odds [29].

This means that the reverse function of the logit function (the sigmoid function $\Phi(\mathrm{z}) = \frac{1}{1+e^{z}}$) can be used to transform every number (i.e. the weighted sum of the features) into a number between 0 and 1 (i.e. the odd ratio of the binary classification):

$$logit\big(p(y=1|x)\big) < w_o * x_0 + w_1 * x_1 + \cdots + w_p * w_p * b$$

Eq. 4-6 Calculation of the probability of a features set (x) being part of class y. w are the weights for the respective features and b is the intercept [29].

The outcome of the sigmoid function – the predicted probability – can then be converted into a binary outcome via a step function:

$$\hat{y} = \begin{cases} 1\ if\ \Phi(\mathrm{z}) \geq 0.5 \\ 0\ if\ \Phi(\mathrm{z}) < 0.5 \end{cases} = \begin{cases} 1\ if\ \mathrm{z} \geq 0 \\ 0\ if\ \mathrm{z} < 0 \end{cases}$$

Eq. 4-7 Step function for binary classification. Compare Figure 4-2 z and Φ(z).

Within the logistic regression, a gradient descend based algorithm is used to minimize the costs, i.e. the difference between predicted probability and true class. Because of this, the model does not

only penalize wrong predictions, but also the certainty of the wrong classification respectively the uncertainty of true classifications [29].
The LR model in its basic form can be used for binary classification, whereas the more sophisticated 'One vs Rest' (OvR) technique allows to use LR for multi-classification tasks. In this case the classification process is split into c sub classifications, where c is the number of classes. Every class has its 'own' model predicting the probability whether a new data point is within the class or not. The model predicting the highest probability for the new data point determines its class [29].
Linear models generally tend to overfit in high dimensions. This is expressed in two ways. Firstly, the model is not able to generalize well, i.e. predict unknown data correctly, although good training results are obtained. Secondly, the model becomes complex and difficult to interpret. The reason for this overfitting lies within the potential/power of LR to find the correct set of weights to even classify outliers and noise correctly. The chance of overfitting increases with the number of dimensions [28].
These effects can be controlled via a regulating term within the loss function. The regulation adds a penalty term to the cost function for each weight. This adds a tradeoff between correct classification of a training point and the length of the weights vector ('L2'-regulation) [29]. This leads to smaller weights and consequently to a simpler model with better generalization quality, although there is a trade-off between model simplicity and generalization potential. The size of the penalty (λ) is set via the parameter c, which is the reciprocal value of λ [29]. Table 8 lists the hyperparameters and preprocessing steps used during the optimization process of the LR.

Table 8 The parameter grid used to optimize the logistic regression .

Parameter	Values
Polynomial Features	[Yes, No]
PCA (components) (normal/poly)	[10,20,30,40,50,60,70,80,90,100,119] [10,100,500,1000,2000]
penalty	[L1, L2]
C	[10^{-4},…, 10^{4}]

4.6 Support Vector Classifier

Support Vector Machines (SVM) are another linear model used for classification. In contrast to LR, where the objective function is to minimize the cost function, the SVM aims to maximize the margin [29]. The margin is defined as the distance between the separating line respectively hyperplane (i.e. decision boundary) and the closest training samples [46]. These training samples are the so-called support vectors. The calibration of the decision boundary is based only on those support vectors. Samples that are further away from the boundary do not influence the slope or position of the hyperplane. The underlying assumption is that a maximal separating hyperplane is a good method

of generalization and the chance of overfitting and the negative effect of outliers is thereby reduced [29].
An important concept for the SVM is the slack variable ξ introduced by *Vladimir Vapnik* [46]. The slack variable allows convergence of linear algorithms, although the data is not linearly separable. Misclassified training samples are penalized accordingly. This penalty resembles the function of the L2 classification used during the logistic regression (4.5) and is also adjusted via the parameter C.
Large values for C lead to a high penalty for the misclassified samples and therefore increase the chance of overfitting. Small values increase the weight of the maximal margin in the objective function. The first part of Eq. 4-8 expresses the maximum margin w, the second the sum of all slack variables for every misclassified samples times C [29].

$$\min = \frac{1}{2}\left|\left|w\right|\right|^2 + C * \sum_{i=1}^{i} \xi^{(i)}$$

Eq. 4-8 Objective function for SVM [29].

The training data is not perfectly linearly separable. Thus, the prediction quality for the linear models is reduced. SVM can solve nonlinear problems by kernelizing the data. The idea behind kernel methods is to create nonlinear combinations of the original features to project them onto a higher dimensional space via a mapping function (Eq. 4-9), where the data becomes linearly separable [29].

$$F(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$$

Eq. 4-9 Example of a simplified mapping function to transform a two-dimensional feature set into a three-dimensional feature set [29].

This mapping function be an additional preprocessing step that is semi-automatically performed by the SVC. It is called 'kernel trick' [28], [29], [46]. There are two common ways to map the input data into a higher dimension: the **polynomial kernel** and the **radial bias function**. The polynomial kernel computes all polynomials of the features up to a threshold. Radial bias function (rbf) is also known as the 'Gaussian kernel'. *Guido* and *Müller* (2016) [28]summarize the functional principle as following:

> *'One way to explain the Gaussian kernel is that it considers all possible polynomials of all degrees, but the importance of the features decreases for each additional dimension.'*

One big disadvantage of SVM is its reduced scalability in terms of sample size. The Sklearn documentation for SVC mentions an upper limit of 10^5 samples, compared to the training data set which includes 4.3 *10^5 samples. Within the constructed model, one approach to the solution of this problem is to trim the data and to only use the last 8 760 values corresponding to the year 2014. While optimizing an SVC there are three specific parameters to optimize. C, the kernel type, and a kernel specific parameter. Within my model I investigated three different kernels: 'linear'-kernel limits the SVC to a linear separation and has no additional parameters. 'Polynomial' (poly)

introduces polynomials of the features as kernel function. The controlling parameter (degree) sets the threshold for the degrees. A high value for the controlling parameter increases the potential to correctly learn nonlinear relations and simultaneously increases the complexity, computational expense, and the chance of overfitting.
The regulating parameter for the rbf is called 'gamma'. It controls the width of the gaussian kernel and therefore the number of samples considered. A high value for gamma increases the chance to overfit [29]. Table 9 lists the hyperparameters and preprocessing steps used during the optimization process of the SVC:

Table 9 The parameter grid used to optimize the SVC.

Parameter	Values
Polynomial Features	[Yes, No]
PCA (components) (normal/poly)	[10,20,30,40,50,60,70,80,90,100,119] [10,100,500,1000,2000]
C	$[10^{-4},...,10^{4}]$
Kernel	['linear', 'ploy', 'rbf']
Degree(poly)	[2,3,4,5]
Gamma(rbf)	$[10^{-5},\ldots,10]$

4.7 Neuronal Nets/Multilayer Perceptron (MLP)

Neuronal nets (also known as 'deep learning') is a group of MLAs based on the concept of perceptron which was developed 1957 by *Frank Rosenblatt*. The underlying motivation was the attempt to recreate the functionality of the neuron in the brain in order improve the understanding of the biological learning process. The perceptron, the building block of a neuronal network, resembles the biological neuron. Today it is known that the functionality of biological neurons is way more complex, nevertheless the perceptron as a method of automated classification is still used.

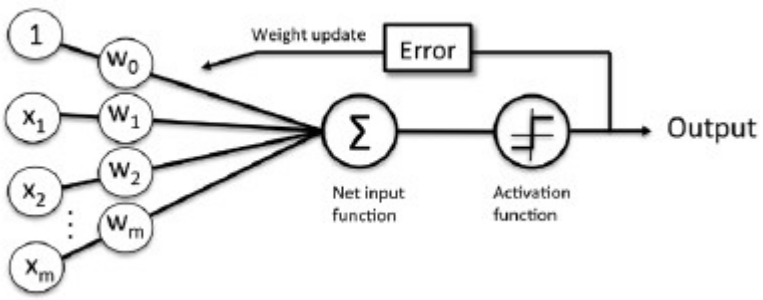


Figure 4-3 schematic illustration of a perceptron for an input vector with m features [25].

The process can be described as follows: A signal (feature) enters a node. Within the node the signal is multiplied by a factor (weights) and an activation function (originally a step function) is applied to transform the result to a binary value. If the classification is correct the weights remain unchanged. Otherwise the weights get updated. If the data is linearly separable, the perceptron will converge [29]. During the last ~70 years the design of the perceptron got improved in various ways (e.g. selection of the activation function).
A modern neuronal net can be described as a multiple parallelly and serially connected perceptron. In its basic form, the feed forward neuronal network is also referred to as 'multilayer perceptron' [28]. Typically, the step function as activation function is replaced by a sigmoid or rectified linear unit (ReLU).

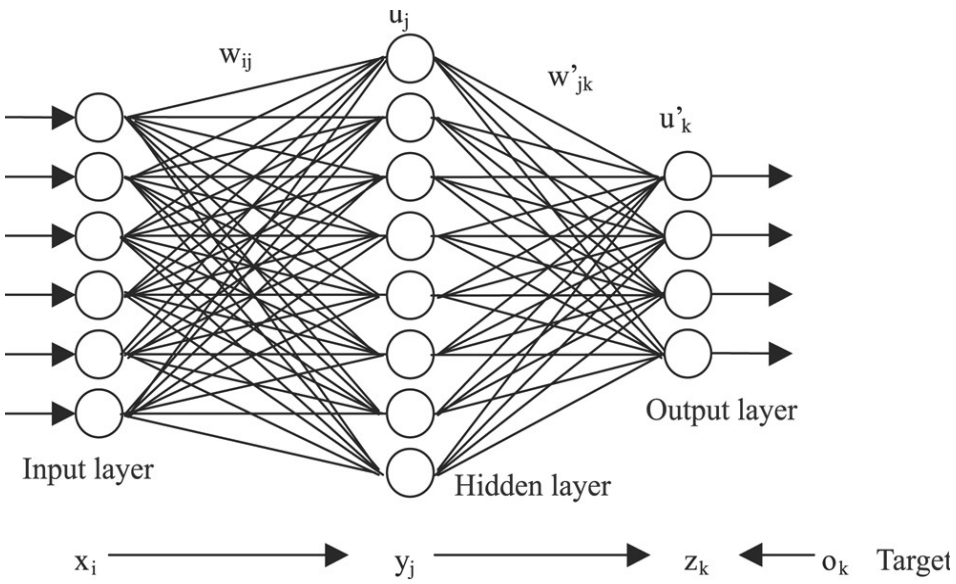


Figure 4-4 schematic illustration of an MLP with an input layer with 6 nodes, one hidden layer with 8 nodes and one output layer with four nodes [46].

Figure 4-4 Illustrates an arbitrary 'fully connected' feed forward neuronal network. The network consists of three layers: one input layer, one hidden layer and one output layer. 'Fully connected' signifies that all nodes i of a layer k are 'connected' to all nodes j of the subsequent layer via the weights vector $w_{ij.}$ This systematic is propagated until the last layer is connected [29].
The prediction process of an MLP is also analog to the algorithm of the perceptron. The output of a node is calculated by applying an activation function on the weighted sums of its input nodes. Typically, a constant value (bias) is added for every node. This process is called 'forward propagation'. The values in the output layer can then be used for the classification. For classification purposes this is typically realized by an 'one-hot' array. During the learning process the weights and biases get updated according to a cost function starting from the output layer. Eventually, the model finds a set of weights and biases able to classify correctly. This process is computationally expensive. Therefore, an appropriate cross-validation becomes impossible. Additionally, MLPs have many

parameters to tune [29], [47]. A random search approach was chosen to find a suitable initial set of parameters. The resulting model was then further optimized gradually by hand.

The dimensions of the input and output layer are predefined by the presentation of the problem: the number of input nodes equals the number of features and the number of output nodes equals the number of classes. Numbers and dimensions of the hidden layers are independent, the dimensions can be optimized, and an adequate activation function must be chosen. A parameter can adjust the learning behavior while the learning rate defines the size of the adjustments of the weights and biases. Large learning rates may overshoot the optimal value, small learning rates may converge to local minima.
Within the current model a dynamic approach was chosen to avoid overshooting or converging behavior. The initial learning rate is set to a large value. If the improvement of the model stagnates the learning rate is decreased. The MLP was realized with the 'keras' framework

5 Results

The optimal hyper parameters for all tested MLA found during the optimization process are shown in Appendix i.a.

5.1 Evaluation Results

Table 10 Training and Evaluation results for the OSA time series of the LP model, the alternative back to back strategies (shift week and shift day) and all classifiers for the training period (2010-2014) and the evaluation period (2015).

Classifier	Grid search CV-results	F1-score (train)	F1-score (test)	profit (test)	relative profit (2015)	number of load cycles	Profit per load cycle
Unit				[EUR]		[n]	[EUR]
Relevant year		2014	2015	2010-14	2015	2015	2015
LP - Optimized (OSA)	-	1.00	1.00	7735.04	1.00	519.00	14.90
Shift Week	-	0.70	0.67	5421.21	0.70	519.00	10.45
Shift Day	-	0.68	0.65	4880.65	0.63	519.00	9.40
K- Nearest Neighbor[22]	0.77	1.00	0.75	5664.58	0.73	400.61	14.14
Decision Tree	0.75	0.85	0.70	5030.10	0.65	451.94	11.13
Random Forest	0.82	1.00	0.77	5302.65	0.69	336.47	**15.76**
Logistic Regression	0.79	0.83	0.77	6038.16	0.78	444.37	13.59
SCV - Linear[23]	0.78	0.80	0.75	5118.63	0.66	437.87	11.69
SVC - RBF[24]	0.81	0.73	0.74	5110.04	0.66	370.78	13.78
Neuronal Network	0.83[25]	0.88	**0.79**	**6116.14**	**0.79**	449.62	13.60

The classifiers are trained on the whole training data set (2010-2014) and are adjusted according to the results of the hyperparameter optimization (Appendix i.a). The scoring results differ slightly from the results of the hyperparameter optimization process. This is partly due to the different data availability in the evaluation processes. During cross-validation, test and training samples are chosen randomly and therefore may lie between two training samples. While training on the data 2010-2014 to predict the 2015 labels, this effect stays out. Table 10 summarizes the results.
'LP-optimized' (OSA) represents the results of the LP model. The OSA set provides the labels for the classification process for the whole period. Therefore, the f1 scores are 1 (or 100%) for the training as well as the testing period.

22 These classifiers were solely trained on the data of 2014.

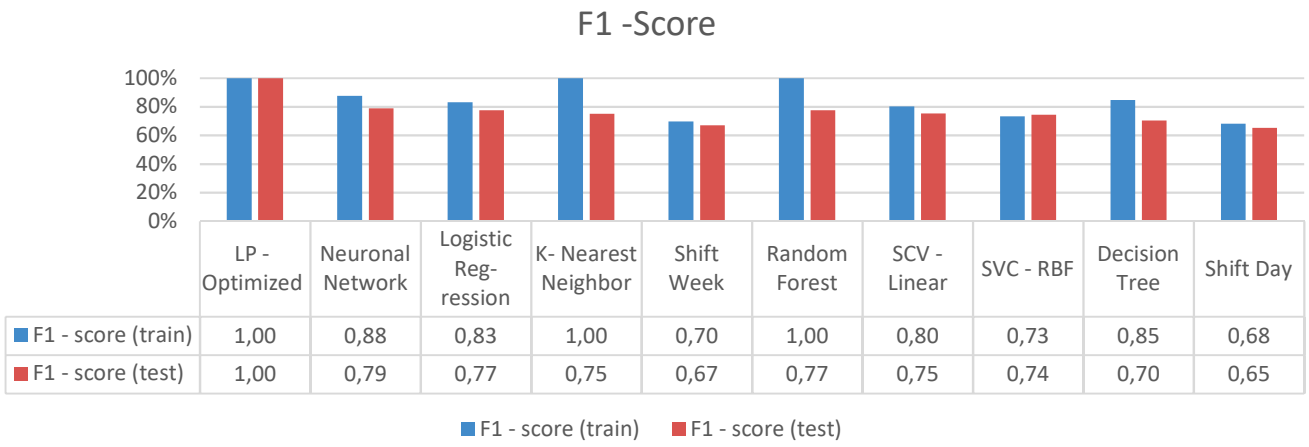
23 These classifiers were solely trained on the data of 2014.

24 These classifiers were solely trained on the data of 2014.

25 Random Search.

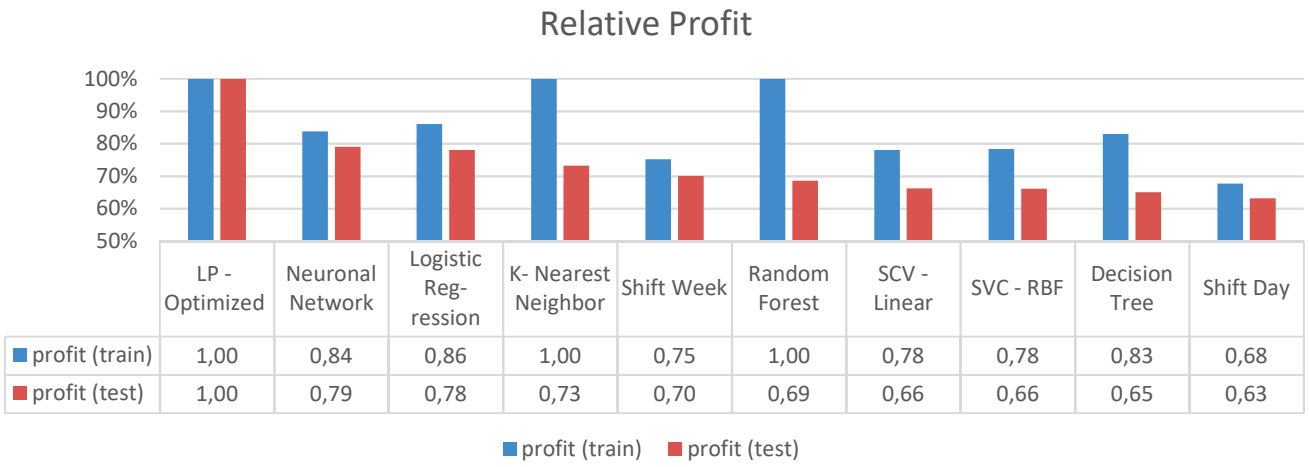
The column ‘profit 2015’ represents the profit that an agent trading based on the signal (e.g. classifier prediction) earns. The relative profit puts the profit into relation of the optimal profit earned by the OSA.
The number of load cycles (LC) is calculated according to Eq. 2-6 and the profit per load cycle is the quotient of the profit and LC. As expected, the OSA has the best performance. With 519 LC it triggers the most complete LC, although the profit per LC is only the second highest.
The quality signal of the back-to-back strategies ‘shift (day)’ and ‘shift week’ can also be expressed by the f1 -score metric. This reveals that both the train and the test scores of ‘shift day’ and ‘shift-week’ (i.e. the back to back strategies) are the worst of the tested methods.
With a relative profit of 70% the shift (week) method outperforms the shift (day) method clearly. The number of LC is naturally similar to the number of LC of the OSA. The profit for LC drops accordingly.
The KNN classifier’s f1 test score is 0.75. Due to the nature of the algorithm, KNN with distance-based voting weights naturally have a f1-score of 1. However, the relative profit of 2015 shows that this classifier outperforms the back- to- back strategies.
The decision tree shows the worst performance of all MLA classifiers tested but still outperforms the shift (day) method. For the ensemble of decision trees, the random forest, two remarkable phenomena are manifesting. A large difference between test f1 score and train f1 score normally indicates an overfit classifier. Usually random forests are more resilient against overfitting than decision trees [28]. However, the hyperparameters were selected based on the best results for f1 determined by a grid search [26]. Interestingly, the random forest shows the lowest number of LCs and the highest profit per LC.
The LR reaches the second-best result. Both support vector machine-based algorithms, although reaching promising qualities of classification, cannot outperform the shift (week) method. The neuronal net (Multilayer Perceptron) reaches the best results while also reaching the highest f1 test score. An agent trading based on the signals of this classifier earns 79% of the optimal profit. The neuronal network also had the highest f1 score during the hyper-parameter optimization. However, the correlation between f1 score and earned profit is not complete.
Figure 5-1 and Figure 5-2 are visualizing the results of the evaluation by the storage logic and the respective relative profits per LC.

26 The analysis of the cross validated grid search log showed, that the maximal f1 test score was reached by an overfitting random forest, see chapter3.4.2



	LP - Optimized	Neuronal Network	Logistic Reg-ression	K- Nearest Neighbor	Shift Week	Random Forest	SCV - Linear	SVC - RBF	Decision Tree	Shift Day
F1 - score (train)	1,00	0,88	0,83	1,00	0,70	1,00	0,80	0,73	0,85	0,68
F1 - score (test)	1,00	0,79	0,77	0,75	0,67	0,77	0,75	0,74	0,70	0,65

Figure 5-1 F1 -scores for all classifiers, shift(week/day) and OSA for the training period (2014) and test period (2015).



	LP - Optimized	Neuronal Network	Logistic Reg-ression	K- Nearest Neighbor	Shift Week	Random Forest	SCV - Linear	SVC - RBF	Decision Tree	Shift Day
profit (train)	1,00	0,84	0,86	1,00	0,75	1,00	0,78	0,78	0,83	0,68
profit (test)	1,00	0,79	0,78	0,73	0,70	0,69	0,66	0,66	0,65	0,63

Figure 5-2 relative profits for all classifiers, shift(week/day) based on the OSA and the evaluation framework for the training period (2014) and test period (2015).

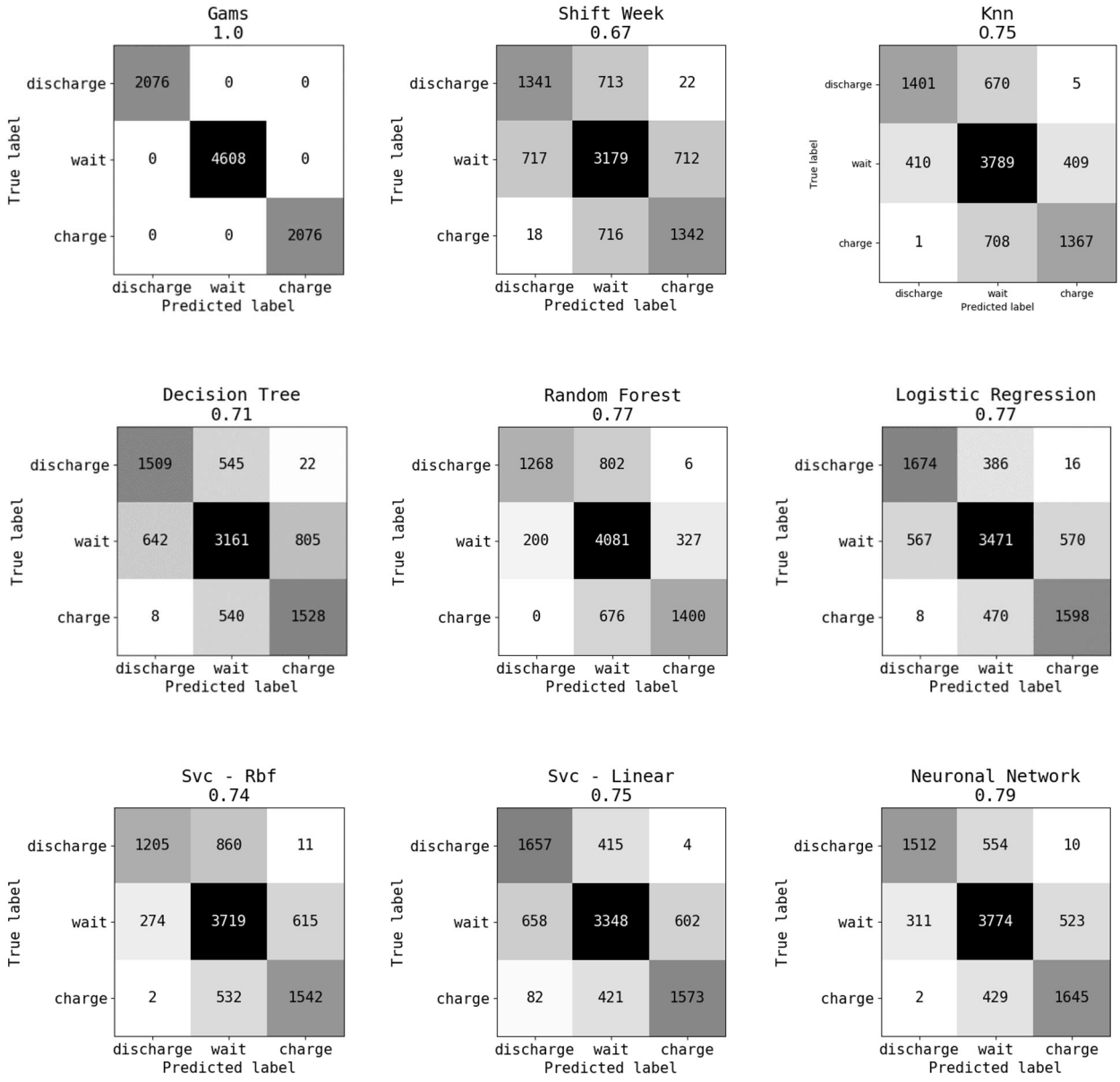


Figure 5-3 Confusion matrices of the the OSA the cLassifiers and the shift(week/day) method. y-axis are the label, x-axis are the predicted labels. The number beneath the title is the f1 score for of the signal.

The calculation of the f1-score is based on the confusion matrix. Confusion matrices for all signals are illustrated in Figure 5-3. Each subplot shows the three by three confusion matrix for one classifier where the y-axis represents the true labels and the x-axis the predicted labels. The numbers in the boxes represent the number of occurrences of a combination, the number below the title is the corresponding f1-score.

At first glance all nine sub figures look very similar but still allow to draw conclusions on closer examination. For example, comparing the results for the RF (2,2) and the LR (2,3): Both have similar f1-scores but completely different profits. The random forest more often correctly predicts ‘wait’

and has clearly less capital errors (predicting the opposite of the true label). The logistic regression however more often classifies the charging and discharging correctly. Overall the logistic regression is more profitable. It can be deducted that the f1-score does not completely correlate with the profit of an agent.

5.2 Visualization of the Different Storage Strategies

The optimal trading strategy calculated by the LPM (the OSA) is based on the diurnal price fluctuations. Throughout the day the average storage pattern shows two charging periods, where electricity is (usually) bought from the market. The first period occurs during the night hours, the second at noon. The discharging periods are usually during the morning hours and afternoon. This pattern resembles typical load profiles. Figure 5-4 illustrates the distribution of the OSA throughout the day. The annotated number represents the count of times the LPM calculated[27] a signal at the corresponding hour. The conditional coloring shows this distinctive pattern and reveals that there are hardly any combinations of a signal and hours with zero occurrences. Additionally, most hours show an occurrence of two signals. This indicates that rigid storage operation models which are solely based on the diurnal pattern are not suitable. It also underlines that the electricity price is influenced by other factors than season or hour of day.

Figure 5-5 to Figure 5-8 display the same figure for the predicted signals by four selected classifiers. It can be seen that all four can reproduce the diurnal trading pattern of the OSA. A closer examination reveals that the patterns of the MLA diverge from the patterns of LPM. While the results of the LPM are less ridged, the MLAs have more hour-signal combinations with zero occurrences. This may indicate prohibitive 'rules' that the model extracts from the training data. (e.g. if it is 18:00: *under no circumstances* buy electricity). An EES operated by such strict rules loses the ability to gain profit in diverging extreme situations, as the LPM would calculate. However, the existence of strict rules doesn't necessarily mean reduced economic performance compared to a more dynamic classifiers' approach. It appears that the RF classifier developed a rather static method of predicting the storage activity based on the time. 'wait' is more common compared to other classifiers and at many points during the day the classifier never predicts a certain option. Between 20:00 and 22:00 it only predicted 'wait'. Compared to the more dynamic strategy of the linear SVC, which predicted more signal-hour combinations and performed worse in means of f1-score and profit than the RF classifier. A deeper analysis of the single strategies developed by the classifier, their particularities, similarities, and the resulting effects on the decision process would lead to a deeper understanding of the decision process and could reveal additional potential for model improvement.

27 "Predicted" in the case of classifiers.

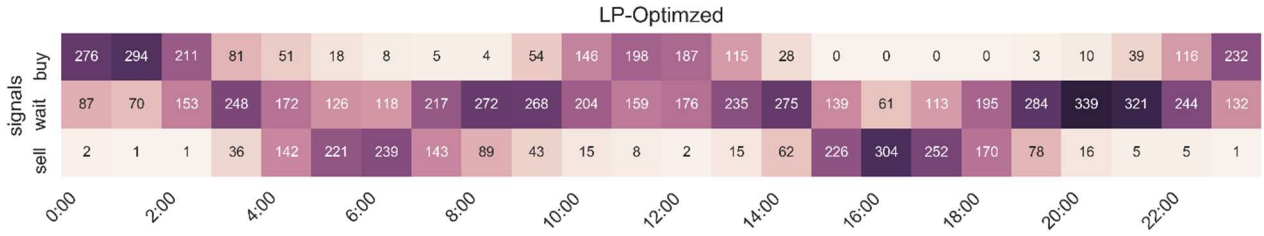


Figure 5-4 Heatmap illustrating the number of signal occurences during the day within the evaluation period 2015 for the LPM optimzed storage behaviour. The time is displayed in GMT.

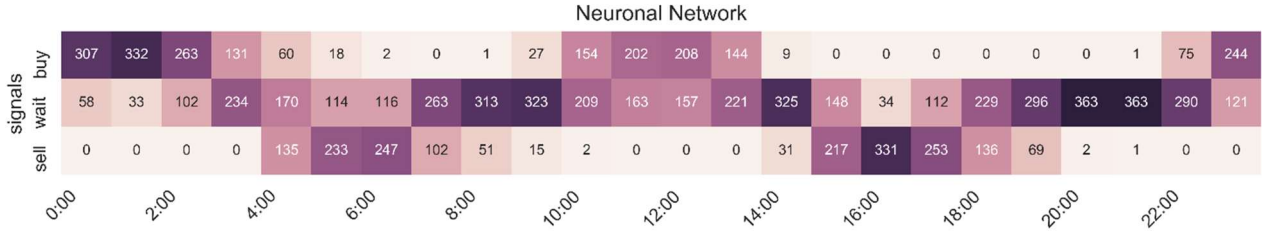


Figure 5-5 Heatmap illustrating the number of signal occurences during the day within the evaluation period 2015 predicted by the Neuronal Network. The time is displayed in GMT.

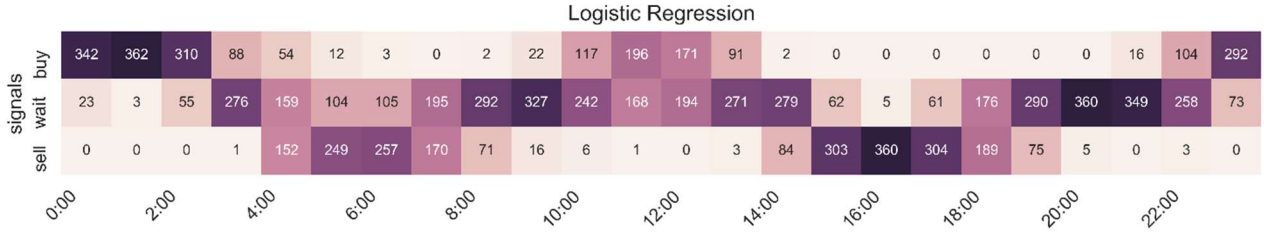


Figure 5-6 Heatmap illustrating the number of signal occurences during the day within the evaluation period 2015 predicted by the Logistic Regression. The time is displayed in GMT.

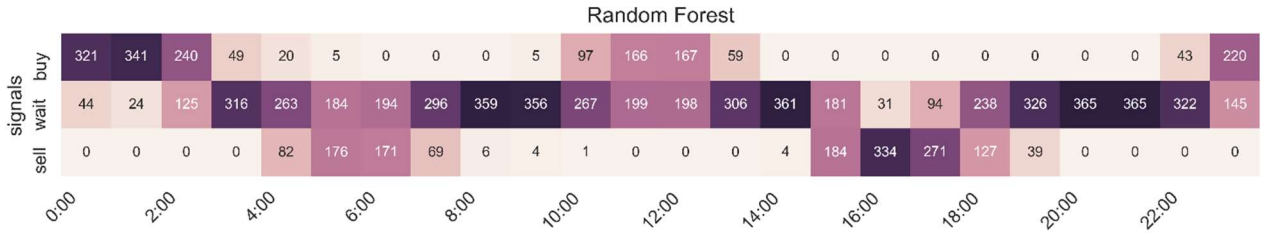


Figure 5-7 Heatmap illustrating the number of signal occurences during the day within the evaluation period 2015 predicted by the Random Forest. The time is displayed in GMT.

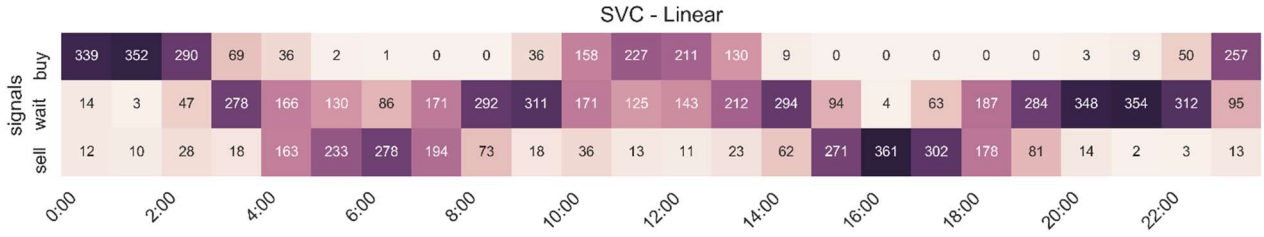


Figure 5-8 Heatmap illustrating the number of signal occurences during the day within the evaluation period 2015 predicted by the SVC. The time is displayed in GMT.

5.3 Correlation between F1-Score and Earned Profit

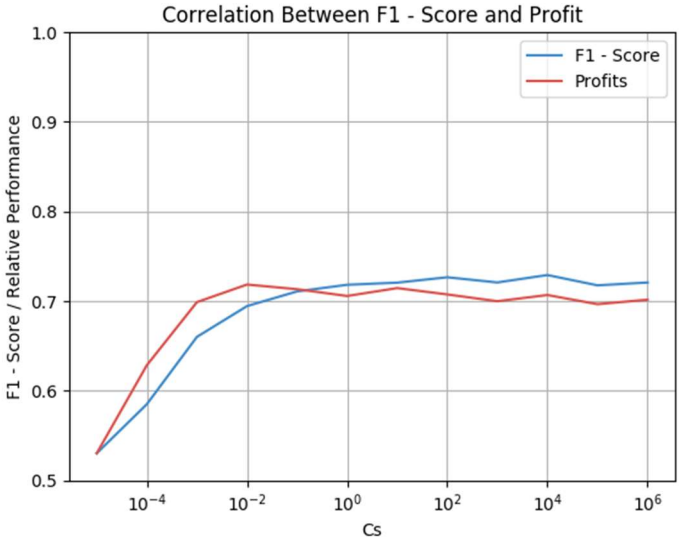


Figure 5-9 The x-axes represents F1 - Score and the simulated relative profits compared to the optimal soloution of logistic regresions with different C values.

The correlation between the maximal profit and the scoring metric during the optimization process cannot be assumed automatically (compare Figure 5-3). A limited correlation hinders the deployment of a classifier with optimal hyperparameters and therefore the maximal profit. The classifier is not capable of learning how bad an error is in economic terms because the metric accounts only for misclassifications. A fictive operator's additional cost of a misclassification cannot be described by the difference in the performance of a single action. The OSA as an agent executes every possible profitable trade. These trades however differ by their profit margin. The development of a specialized metric, calculating the exact costs of misclassification based on complete trades instead of the f1-score, requires a stepwise chronological correct evaluation during the training process. Figure 5-9 shows the correlation of profit and the f1-score for a logistic regression for different values for the c parameter trained on the 2014 dataset and evaluated on the 2015 data set. Both the profit and the f1-score are calculated based on the testing set. The maximal f1-score is reached for c = 10 000 while the maximal profit is reached for c = -0.01. This example shows that the maxima are indeed not congruent, but the difference is rather small. Therefore, the f1-score be a good approximation of the profit. At the same time, it also shows that the f1-score is not the perfect score metric to maximize the profit. Future research might investigate the effects of other customized score metrics during the optimization process.

5.4 Effect of the Forecast Horizon

A second potential effect on the classification quality is the increasing temporal distance between the training and the test set over the evaluation year. In other words: if one trains a classifier based only on the data of the years 2010-2014, the prediction quality for January 2015 may be better than for December 2015. The hypothesis is that temporal developments and trends are changing the optimal behavior pattern. Consequently, the temporal distance between training period and prediction period should have a negative influence on the quality of the classification. To display

potential quality losses over time, a moving average with a window size of four weeks of the accuracy is calculated as an indicator of short-term prediction quality for all classifiers[28].
A linear model was fit to estimate the intercept and slope for all classifiers. A negative slope would indicate a decrease in the model's prediction quality with increased temporal distance to the training period. The results show that there are no considerable negative slopes. The intercept's slopes, p-values, and R^2 are displayed in Table 11:

Table 11 linear models decribing the correlation between temporal distance and classification quality.

Name	Intercept	Slope	P-Value	R^2
Decision Tree	0.71	-0	0.3306	0.0001
KNN	0.76	-0	$<10^{-5}$	0.0549
Logistic Regression	0.78	-0	$<10^{-5}$	0.0046
Neuronal Network	0.79	-0	$<10^{-5}$	0.0151
Random Forest	0.76	0	$<10^{-5}$	0.056
SVC -RBF	0.73	0	$<10^{-5}$	0.32
SVC Linear	0.73	0	$<10^{-5}$	0.32

Although there are temporal oscillations observable, neither the results from the linear model nor the plot of Figure 5-10 suggest significant signs of erosion of the classification quality within a one-year period. However, this is only valid if the economic framework remains unchanged. Disruptive effects (e.g. policy changes, introduction of new technologies) within the market, leading to different optimal storage behavior, will require a retraining of the classifier.

[28] The results show that the moving average of the accuracy is smoother compared to the actual Boolean values of the correctness of a classification (true or false) for every step

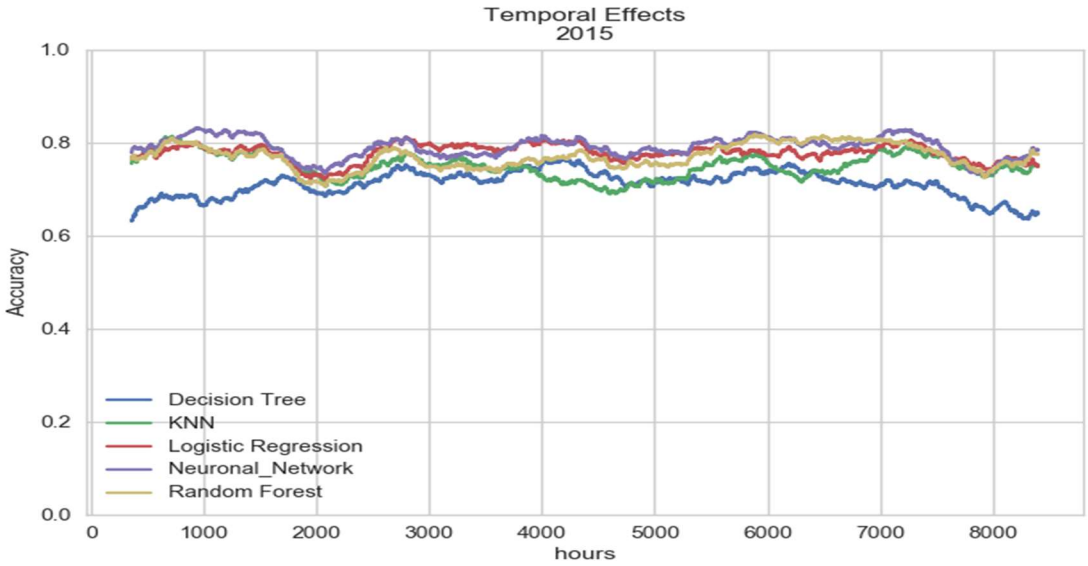


Figure 5-10 Temporal effects on the classification accuracy. Comparision of classifier for the year 2015.

Additionally, Figure 5-10 reveals correlating oscillations within the classification quality. Around the 1 000st hour all classifiers (except the decision tree) are performing better than at the 2 000st hour. These homogenous oscillations indicate that there may be additional influencing factors to consider.

5.5 Rentability of the ESS

The economic rentability is a central precondition for the large-scale market-based deployment of EESs. A mean of calculating this rentability is the net present value (NPV) [4]. Assuming investment costs [C_O] of 300-800 EUR/kWh, a typical lifetime [L_y] for Li-ion accumulators of 5-15 years [25] and a typical discount rate [r] of 10% [4] for investments in the energy sector the NPV of an ESS operated by the algorithm can be calculated according to Eq. 5-1, where C_t represents the annual revenue minus the annual cost for operation and maintenance.

$$NPV = \sum_{t=0}^{L_y} \frac{C_t}{(1-r)^t} - C_o$$

Eq. 5-1 Calculation of the net present value of an investment.

To facilitate the calculation, the annual costs for operation and maintenance are neglected and the annual revenue is assumed to be constant (based on the simulated results for the year 2015). An ESS operated by the predictions of the best performing MLA, the NN, yields a profit of 6 116 EUR per year. Under the most optimistic assumptions of 15 years lifetime and investment costs of 300 EUR/kWh the resulting NPV is -253 480 EUR. To be profitable, the investment costs must be reduced below the threshold of 46.51 EUR/kWh. Considering the optimal solution of the LPM similar results apply. An ESS operated by the LPM yields a profit of 7 735 EUR per year. Resulting in an NPV of -241 166. To be profitable, the investment costs have to be reduced below the threshold of 58.1

EUR/kWh. This shows that the investment costs are yet too high, respectively the annual profits are too low to profitably arbitrage at the German power market with Li-ion batteries.

6 Discussion and Conclusion

6.1 Improving the Model's Quality

The results of the evaluation process show that basic machine learning algorithms can outperform standard strategies like the back to back algorithm. However, compared to typical results for machine learning based classification (accuracy >90%), an accuracy of +/- 80% is tenuous [29], [46]. To increase the classification quality and therefore the economic performance, there are several paths to take. All of them represent different starting points for future research.

6.1.1 Training Process

6.1.1.1 Scaling

In this thesis the possible scaling options are limited to two methods (see chapter 3.3.2). The standard scaling method (mean zero and standard deviation 1) is a minimum requirement for many machine algorithms. However, it is prone to outliers.
The quantile transformer on the other hand maps all values uniformly distributed between -1 and 1 and is therefore robust against outliers. However, the actual set of scaling options to choose from is larger. While many MLAs are designed for feature values with a mean close to zero and a standard deviation close to 1, the method of transformation is arbitrary. Including other scaling techniques (like Normalizing, Robust Scalers) into the hyperparameter tuning process could lead to better results.

6.1.1.2 Feature Selection

Same accounts for feature selection and feature enrichment. In this thesis, the feature enrichment option was limited to a second-degree polynomial feature combination and principal component analysis' number of components. Analog to the scaling step, there are additional techniques like Linear Discriminant Analysis or Kernelized PCA to be considered during the optimization process [29].

6.1.2 Additional Data

Another typical method of increasing a model's performance is the gathering of additional data. In general, this includes additional samples as well as the addition of feature dimensions. Since the economic conditions for energy storage vary over time, additional samples with increased temporal distance to the prediction period won't necessarily increase the classification potential of the model. An additional feature, representing factors like weather conditions, holidays or daylight-saving time could improve the classification quality [48].

6.1.3 Economic Performance

A method to increase the economic performance of the model rather than the classification quality is based on the decision boundaries. All classifiers classify based on a probability value (e.g. the percentage of votes per class). This value is a continuous number. By default, the category with the highest probability value gets predicted. Tuning these values can improve the performance of the agent.
Within the prediction process a trained KNN classifier returns an array of values. Each value represents the percentage of votes a certain class gets. By default, the class with the most votes and therefore the highest value determines the class. Yet, this decision process can be modified. With the three classes of the EES in mind (charge, discharge, wait), an alternative strategy determining the prediction process could be as following: at least 90% of the neighbors must vote either charge or discharge for the ESS to take action. This would represent a conservative strategy, avoiding false actions, while accepting more downtime. An analysis based on a simple algorithm that shows that further improvements of the classifiers results are realizable can be found in Appendix i.c

6.2 Shortcomings of the Model

The model in its current state, regarding extension, prediction quality and flexibility, shows the basic feasibility of EES operation based on MLAs. At the same time, it illustrates some of the shortcomings, that must be overcome to successfully implement MLAs as operation algorithms. The forecast horizon is currently limited to one hour which reduces the applicability of the model dramatically. Especially as the trading rules of the EM hinder live/online trading. Principally the model could also learn to predict arbitrary time steps into the future (i.e. +12 hours). Since the lagged variables seem to play an important role for the prediction process (see 'back to back' and 0) the prediction quality should remain acceptable. A series of models, each predicting an hour in the future (i.e.+1, +2 ...,+24 hours), can be used jointly for more complex predictions for every timestep.

Secondly, the model in its current state makes use of only one possible revenue stream (arbitrage trading). As already elaborated in chapter 2, an EES potentially offers multiple services (i.e. regulating energy, intraday market) to the EM. Under real market conditions a profit orientated EES operator wants to provide a mix of services to gain maximum profit for her facility. The linear optimization of a storage problem for several markets respectively services is not trivial. Simultaneously, the approach I chose requires an optimal (or at least very good) solution to classify by in order to obtain satisfying results. Additionally, the current model has only three classes (alternatives) to 'choose' from and therefore a rather simple task to learn. For more complex models the underlying decisions to be made by the operator, may exceed the MLAs 'learning' capacities . Anyway, the problem regarding the evaluation of classification (second best to worst alternative, see chapter 3.5.1) exaggerates because of the additional classes/alternatives.

Thirdly, a large-scale deployment of EESs with same or similar trading algorithms increases the impact of the trading process on the price building process. Consequently, the 'pricetaker' assumption during the linear optimization process becomes obsolete. This limits the maximal profits of the EES and may lead to unexpected price deviations. However, even for large scale deployments, EESs still provide balancing services for the market. This may lead to a shift in in the weighting of the single input features of the model(s) (e.g. a strategy change). A constant evaluation of the model, its features, and its profitability is therefore necessary.

6.3 Final Summary

To evaluate the potential of machine learning based algorithms for electricity storage control (MLAES) a programming framework was introduced. Within this framework, the potential performance of different algorithms was compared. The results show that MLAES can outperform simple storage strategies and reduce the gap to the mathematically optimal solution. There is potential for further improvement of the model quality and the potential profit. This proves that MLAES are feasible for storage operation and should be investigated further. As the assumptions made in this thesis lead to a simplification, the results can't be transferred one-to-one to real-world conditions.

Although MLAES are shown to outperform other strategies such as back to back trading, the deployment of EES for arbitraging a single EM is not profitable now. The high investment costs prohibit the profit-oriented deployment, which also includes the mathematically optimal results of the linear model.

Decreasing investment costs and increased political efforts to introduce EES into the market might change this. At the current market situation, it cannot be expected that EES are deployed solely for arbitraging on electricity wholesale markets. Similar results can be found in other publications [49], [50].

7 Literature

[1] UNFCCC, “Paris Agreement,” *Conf. Parties its twenty-first Sess.*, vol. 21932, no. December, p. 32, 2015.

[2] J. Rockström, O. Gaffney, J. Rogelj, M. Meinshausen, N. Nakicenovic, and H. J. Schellnhuber, “A roadmap for rapid decarbonization,” *Science (80-.).*, vol. 355, no. 6331, pp. 1269–1271, 2017.

[3] A. J. Schwab, *Elektroenergiesysteme*, vol. 53, no. 9. 2012.

[4] M. Sterner and I. Stadler, *Energie Speicher*, vol. 53. 2013.

[5] Joachim Nitsch *et al.*, “Langfristszenarien und Strategien für den Ausbau der Erneuerbaren Energien in Deutschland bei Berücksichtigung der Entwicklung in Europa und global. Schlussbericht BMU - FKZ 03MAP146,” Bonn, 2012.

[6] B. W. Schill, J. Diekmann, and A. Zerrahn, “Power Storage: An Important Option for the German Energy Transition,” vol. 3, no. 3, pp. 137–147, 2015.

[7] G. Fuchs, B. Lunz, M. Leuthold, and D. U. Sauer, “Technology Overview on Electricity Storage - Overview on the potential and on the deployment perpectives of electric storage technologies,” *Inst. Power Electron. Electr. Drives (ISEA), RWTH Aachen Univ.*, no. June, p. 66, 2012.

[8] VDE Association of Electrical Electronic and Information Technologies, “The German Roadmap: Smart Grid, Standardization, Status, Trends and Prospects,” 2013.

[9] Frauenhoferinstitut, “Net generation of power plants for public power supply.,” 2018. [Online]. Available: https://www.energy-charts.de/energy_pie.htm?year=2017. [Accessed: 10-Mar-2018].

[10] M. Nicolosi, “Leitstudie Strommarkt 2015,” no. final report, pp. 1–99, 2015.

[11] D. R. Graeber, *Handel mit Strom aus erneuerbaren Energien*. wiesbaden: springer, 2014.

[12] D. McConnell, T. Forcey, and M. Sandiford, “Estimating the value of electricity storage in an energy-only wholesale market,” *Appl. Energy*, vol. 159, pp. 422–432, 2015.

[13] B. Zakeri and S. Syri, “Electrical energy storage systems: A comparative life cycle cost analysis,” *Renew. Sustain. Energy Rev.*, vol. 42, pp. 569–596, 2015.

[14] W.-P. Schill, “Integration von Wind- und Solarenergie: Flexibles Stromsystem verringert Überschüsse,” *DIW-Wochenbericht*, vol. 80, no. 34, pp. 3–14, 2013.

[15] D. Zafirakis, K. J. Chalvatzis, G. Baiocchi, and G. Daskalakis, “The value of arbitrage for energy storage: Evidence from European electricity markets,” *Appl. Energy*, vol. 184, pp. 971–986, 2016.

[16] W. P. Schill, “Residual load, renewable surplus generation and storage requirements in Germany,” *Energy Policy*, vol. 73, pp. 65–79, 2014.

[17] M. B. C. Salles, M. J. Aziz, and W. W. Hogan, “Potential arbitrage revenue of energy storage systems in PJM during 2014,” *IEEE Power Energy Soc. Gen. Meet.*, vol. 2016–Novem, 2016.

[18] R. Sioshansi, P. Denholm, T. Jenkin, and J. Weiss, “Estimating the value of electricity storage in PJM: Arbitrage and some welfare effects,” *Energy Econ.*, vol. 31, no. 2, pp. 269–277, 2009.

[19] Y. Yoon and Y. H. Kim, “Charge scheduling of an energy storage system under time-of-use pricing and a demand charge,” *Sci. World J.*, vol. 2014, 2014.

[20] J. Ma and X. Ma, “State-of-the-art forecasting algorithms for microgrids,” *ICAC 2017 - 2017 23rd IEEE Int. Conf. Autom. Comput. Addressing Glob. Challenges through Autom. Comput.*, no. September, pp. 7–8, 2017.

[21] K. Bradbury, L. Pratson, and D. Patiño-Echeverri, “Economic viability of energy storage systems based on price arbitrage potential in real-time U.S. electricity markets,” *Appl. Energy*, vol. 114, pp. 512–519, 2014.

[22] N. Yu and B. Foggo, “Stochastic valuation of energy storage in wholesale power markets,” *Energy Econ.*, vol. 64, pp. 177–185, 2017.

[23] V. Krishnan and T. Das, “Optimal allocation of energy storage in a co-optimized electricity market: Benefits assessment and deriving indicators for economic storage ventures,” *Energy*, vol. 81, pp. 175–188, 2015.

[24] Bloomberg, “New Energy Finance,” 2016. [Online]. Available: https://www.bnef.com/dataview/new-energy-outlook-2016/index.html.

[25] T. Bocklisch, “Hybrid energy storage systems for renewable energy applications,” *Energy Procedia*, vol. 73, pp. 103–111, 2015.

[26] ENTSOE, “European Network of Transmission System Operators for Electricity.” [Online]. Available: https://www.entsoe.eu. [Accessed: 01-Dec-2017].

[27] Open Power Data System, “Time Series,” 2017. [Online]. Available: https://open-power-system-data.org.

[28] S. Guido and A. C. Müller, *introduction to Machine Learning with Python*. O’Reilly Media, 2016.

[29] S. Raschka, *Python Machine Learning*. BIRMINGHAM: Packt Publishing, 2015.

[30] Austrian Power Grid, “apg.at.” [Online]. Available: www.apg.at. [Accessed: 11-Oct-2017].

[31] D. I. Chatzigiannis, G. A. Dourbois, P. N. Biskas, and A. G. Bakirtzis, “European day-ahead electricity market clearing model,” *Electr. Power Syst. Res.*, vol. 140, pp. 225–239, 2016.

[32] R. M. Karp R, “Deorg Danzigs impact on the theory of computation discrete optimization,” *Discret. Optim.*, vol. 5, pp. 174–185, 2008.

[33] C. Lewis, “Linear Programming : Theory and Applications,” p. 65, 2008.

[34] G. James, D. Witen, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with*

Applications in R, vol. 64, no. 9–12. New York: Springer, 2007.

[35] G. Varoquaux, O. Grisel, and R. RV, "sklearn.model_selection.train_test_split," *sklearn*, 2017. [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/model_selection/_split.py#L1920. [Accessed: 20-Dec-2017].

[36] R. RV, G. Lemaitre, and T. Unterthiner, "SKLearn - Compare the effect of different scalers on data with outliers." [Online]. Available: http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html. [Accessed: 23-Nov-2017].

[37] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2012.

[38] A. Gramfort *et al.*, "sklearn.preprocessing.quantile_transform," *sk learn*. [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/preprocessing/data.py#L2450. [Accessed: 23-Nov-2017].

[39] G. Louppe *et al.*, "sklearn.tree.DecisionTreeClassifier." [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier. [Accessed: 30-Nov-2017].

[40] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[41] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning," *Math. Intell.*, vol. 27, no. 2, pp. 83–85, 2001.

[42] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.

[43] P. Flach and M. Kull, "Precision-Recall-Gain Curves: PR Analysis Done Right," *Adv. Neural Inf. Process. Syst. 28*, vol. 1, pp. 838–846, 2015.

[44] J. Vanderplas, "sklearn.neighbors.DistanceMetric," 2013. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html. [Accessed: 29-Nov-2017].

[45] J. Vanderplas, F. Pedregosa, and A. Gramfort, "sklearn.neighbors.KNeighborsClassifier," *sklearn*, 2017. [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/neighbors/classification.py#L23. [Accessed: 20-Dec-2017].

[46] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification," *BJU Int.*, vol. 101, no. 1, pp. 1396–400, 2008.

[47] F. Chollet and others, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras.

[48] G. Angenendt, S. Zurmühlen, R. Mir-Montazeri, D. Magnor, and D. U. Sauer, "Enhancing Battery Lifetime in PV Battery Home Storage System Using Forecast Based Operating Strategies," *Energy Procedia*, vol. 99, no. March, pp. 80–88, 2016.

[49] A. Belderbos, E. Delarue, K. Kessels, and D. William, "The Levelized Cost of Storage

critically analyzed and its intricacies clearly explained The Levelized Cost of Storage critically analyzed and its intricacies clearly explained," no. december, p. 28, 2016.

[50] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future," *Int. J. Forecast.*, vol. 30, no. 4, pp. 1030–1081, 2014.

[51] L. Breiman and C. Adele, "Random Forests." [Online]. Available: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm. [Accessed: 24-Nov-2017].

[52] D. Magnor and D. U. Sauer, "Optimization of PV Battery Systems Using Genetic Algorithms," *Energy Procedia*, vol. 99, no. March, pp. 332–340, 2016.

[53] J. Weniger, T. Tjaden, J. Bergner, and V. Quaschning, "Sizing of Battery Converters for Residential PV Storage Systems," *Energy Procedia*, vol. 99, pp. 3–10, 2016.

Appendix i. **Additional Information**

a. Optimal Hyperparamter

The following tables (Table 12 to Table 17) summarize the optimal parameter selection for each classifier based on their mean f1 score during the 5-fold cross validation for the years 2010 -2014. These values represent the optimal settings for the hyperparameters.

Table 12 Gridsearch resutlts k- nearest neighbors (see chapter 4.2).

Pre-processing/hyperparameter	Value
Scaler	Quantile transformation
Number of neighbors	11
Weights	uniform
Mean CV score (f1)	0.775
Standard deviation of CV score	0.004

Table 13 Gridsearch results decision tree (see chapter 4.3).

Preprocessing/hyperparameter	Value
Impurity metric	"Entropy"
Maximal depth	50
Minimal samples for an additional split	2
Minimal samples in terminal node	1
Mean CV score (f1)	0.746
Standard deviation of CV score	0.005

Table 14 Gridsearch results random forrest (see chapter 4.4).

Preprocessing/hyperparameter	Value
Impurity metric	"Entropy"
Number of trees	100
Maximal Features	90
Mean CV score (f1)	0.821
Standard deviation of CV score	0.005

Table 15 Gridsearch results logistic regression (see chapter 4.5).

Preprocessing/hyperparameter	Value
Scaler	Quantile Transformer
Polynomial Features	Yes
PCA	1000
penalty type	L2
penalty (C)	0.1
Mean CV score (f1)	0.795
Standard deviation of CV score	0.007

Table 16 Gridsearch results support vector classifier (see chapter 4.6).

Preprocessing/hyperparameter	Value
Scaler	Standard Scaler
Polynomial Features	No
Kernel type	RBF
Penalty (C)	100
Gamma	0.001
Mean CV score (f1)	0.810
Standard deviation of CV score	0.006

Table 17 random search neuronal net (see chapter 4.7).

Preprocessing/hyperparameter	Value
Polynomial Features	No
Scaler	Quantile Transformer
input layer	119
input layer activation function	Sigmoid
Number of hidden layer	3
Nodes hidden layer	119
Dropout during Training	10% after each layer
Activation function hidden layer	hard sigmoid (steeper version of sigmoid)
Output nodes	3
Output layer activation function	softmax
Batch size	50
Epochs	50
Optimizer	RMSprop
Initial leaning rate	0.01
Loss function	categorical cross entropy
Test Score	0.83

b. Model based feature selection

Some models (e.g. Random Forest, Logistic Regressions) can be used to estimate the importance of single features. One of them is the random forest model (chapter 4.3). While training the model, the importance of features is determined based on the decisions of the trees. This importance, also called 'Gini-importance' (based on the Gini coefficient) is the normalized total reduction of the Gini coefficient of a population while splitting the population based on this certain feature [39]. In other words, the Gini coefficient shows how much impurity/entropy reduction can be obtained by a (data) split based on this exact feature. Thus, features carrying useful information could be pushed out by chance [29]. This effect can be reduced by repeating the process several times [29], [51]. Figure 7-1. displays the Gini-importance and the potential thresholds for feature selection for the original training data.

Figure 7-2 does the same for the polynomial expanded data (see chapter 3.3.2). It is important to acknowledge that, although the selection is based on 1 000 unique decision trees, the order and therefore the constellation of the selected features varies slightly between each draw.
The table on the left side shows the 10 'most important' features according to the RF model. On the right side all Gini-importances are displayed in descending order. The top 80 features are responsible for 90% of the entropy reduction. Around the 80th feature the additional information gain becomes very small. Features after the 80th are considerable to be dropped. A manual examination of these features shows that these features are without exception containing dummy variables for the hour, weekday, and month.

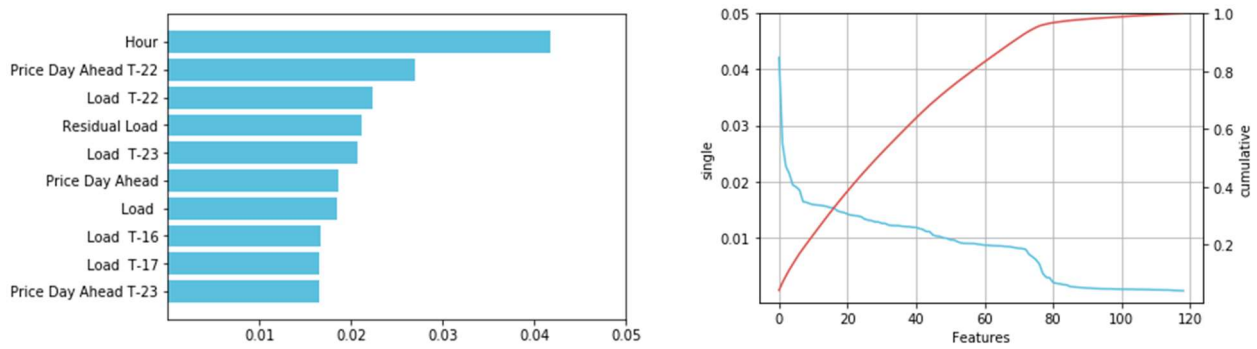


Figure 7-1 Gini importances of the features for the orignial data. Right plot: importances of the single fetures in bule and the cumulative importances in red.

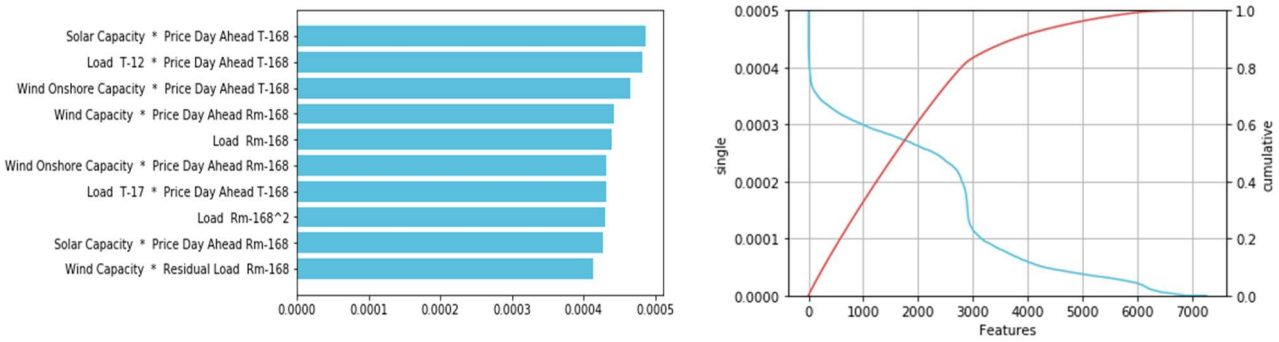


Figure 7-2 Gini-importances of the features for the polynomial enriched training set. Right plot: importances of the single fetures in bule and the cumulative importances in red.

A closer look at the features respectively their combinations reveals interesting results. In this (pseudo random) draw, the installed capacity of wind and solar and the price of T-168 are the dominant features. These combinations and their causal relations are surprising since the installed capacities are rather constant over time. However, these top features are probably highly correlated and therefore the ability to make predictions solely based on them is low.

On the right side all importances are displayed in a descending order. In blue (left axis) the values for each single feature and in red (right axis) the cumulative importance. Note the difference in the scale of the axis between the original and the polynomial dataset.

The Gini-importance is a relative value and therefore decreasing with the total number of features. Two phenomena are visible. Firstly, the last ~500 features add no additional information. These are the columns of zeros. Secondly, the main drop of importance occurs at 3 000 features. For most MLAs datasets containing 3000 dimensions and ~3 0000 rows are too large, considering the computational limitations of an ordinary PC and the standards of conscientious training and test regimes.

c. Performance Improvement via Decision Boundaries

The artificial optimization of the decision boundaries can be achieved by simply iterating over the different thresholds for the decision boundary. In this example, the algorithm is tested for 100 uniformly distributed thresholds between 0 and 1. This results in a new array of predicted storage activities. For every prediction: either the predicted value for charge or discharge greater than the threshold. If yes, predict the higher of those two else predict wait. This results in 100 different classifications to be evaluated. For the same KNN classifier as used during the evaluation the optimal threshold of 0.33 results in 79.5% of the maximal profit for the year 2015. The default setting results in 73%. This means that if either the predicted probability for charge or discharge is greater than 33% the one with the highest probability should be taken or if not, then wait. While optimizing this threshold for the maximal profit for the year 2015 represents an illegal information transfer, the same can be done for the year 2014. Optimizing the decision boundaries for 2014 results with an optimal threshold of 0.30. Appling this threshold (optimized on the year 2014) on the year 2015 the profit still is 79% of the maximal profit. This is only 0.5% less than the optimal result for 2015 from neuronal networks. This example shows that the tuning of the decision boundaries further improves the performance substantially without any additional information. Same applies for all other classifiers.

However, the description of an optimal algorithm, in combination with metrics to determine the best setting of the decision boundaries requires additional research. Figure 7-3 illustrates the effect of the threshold on the profit of the EES. It is visible that the profit in dependency of the threshold correlates for both sets.

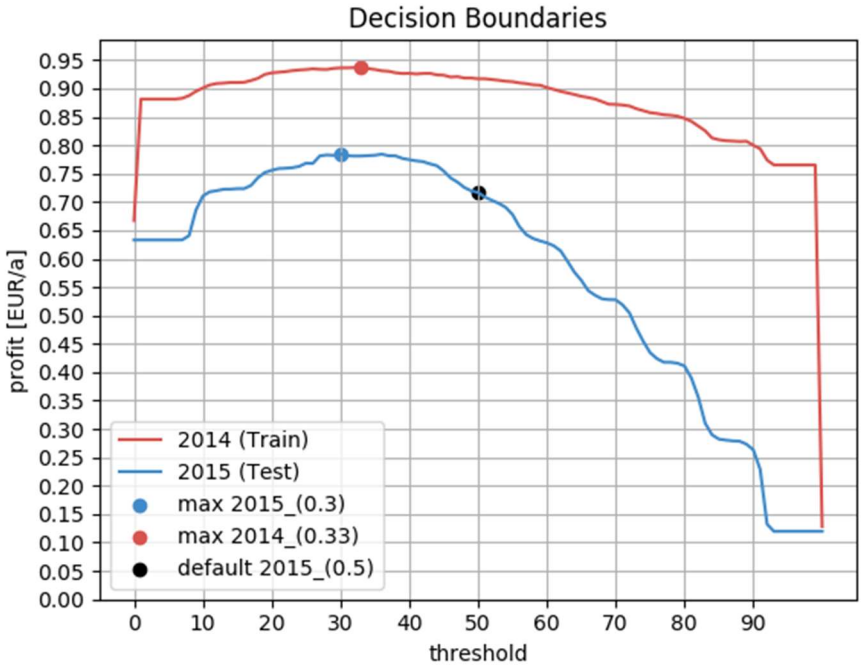


Figure 7-3 Decision Boundaries for the a KNN classifier trained on 2014's data tested on 2015 data. The decision boundaries are based on a threshold for classification.

Appendix ii. **Code**

The complete code is also available at: https://github.com/zwiebo/Evaluation-of-Machine-Learning-Based-Storage-Control-Algorithms-for-the-Electricity-Market

a. Data import & Data Cleaning

```
"""
# This Script manages the data import from Open-Power-System-Data (OPSD)
# the source file is downloaded manually and from the OPSD homepage

# The tasks of the script are:
#   - load the data from a .csv file into a "pandas" DataFrame
#   - transform the time stamp string to a time object
#   - delete unnecessary columns
#   - delete columns with to many empty values
#   - calculate some values of interest
        - residual load
        - combined forecasts
        - combined renewable generations
#   - save the DataFrame to a '.xls' -> OPSD_OWN_table.xls
#   - save a pickle for the next run

"""

from datetime import datetime
import pandas as pd
import warnings

# ---------
# VARIABLES
# ---------

# file import
original_file = "time_series_60min_singleindex_filtered.csv"
df = pd.DataFrame()

# ---------
# Functions
# ---------

def transform_date(time_stamp_str):
    """
    ----------
    # convert the time stamp string from the file to a machine readable object
    ----------
    """
    return datetime.strptime(time_stamp_str, "%Y-%m-%dT%H:%M:%SZ")

def format_workday(value):
    """
    Classify a variable as either workday or weekend-day
    :param value: number of day
    :return: 1 : workday,
             0 : weekendday
    """
    if value <= 5:
        return 1
    else:
```

```
        return 0

def init_time_stamps():
    """
    format the timestamp and make variables for every component of the timestamp
    """
    df["absolute_hour"] = df.index
    df["Time"] = list(map(transform_date, df["utc_timestamp"]))
    df["hour"] = pd.DatetimeIndex(df["Time"]).hour
    df["day"] = pd.DatetimeIndex(df["Time"]).day
    df["month"] = pd.DatetimeIndex(df["Time"]).month
    df["year"] = pd.DatetimeIndex(df["Time"]).year
    df["weekday"] = pd.DatetimeIndex(df["Time"]).weekday
    df["workday"] = list(map(format_workday, df["weekday"]))


def create_header_overview(DataFrame=df, Filename="00_Headers.txt"):
    """
    create a file with all headers of the DataFrame for a better overview
    """

    with open(Filename, "w") as f:
        for i in list(DataFrame):
            f.write("'{}',\n".format(i))


if __name__ == "__main__":
    # load Dataframe
    df = pd.read_csv(original_file, index_col=1, parse_dates=True)
    init_time_stamps()
    # drop columns with lots of NAN
    df.drop('interpolated_values', axis=1, inplace=True)
    df.drop("DE_wind_offshore_generation", axis=1, inplace=True)
    df.drop("DE_wind_onshore_generation", axis=1, inplace=True)
    df.drop("DE_50hertz_wind_offshore_forecast", axis=1, inplace=True)
    df.drop("DE_50hertz_wind_onshore_forecast", axis=1, inplace=True)
    df.drop("DE_50hertz_wind_offshore_generation", axis=1, inplace=True)
    df.drop("DE_50hertz_wind_onshore_generation", axis=1, inplace=True)

    # calculate the residual load
    # take all time steps with wind and solar generation
    # subtract the Solar and PV generation from the total load
    df["DE_solar_generation"] = df["DE_solar_generation"].fillna(value=0)
    warnings.warn("Fill na in 'DE_solar_generation' with '0' !")

    # calculate the total renewable generation
    df["renewable_generation"] = df["DE_wind_generation"] +
df["DE_solar_generation"]
    df["DE_residual_load"] = df["DE_load_"] - df["renewable_generation"]

    # calculate the total wind forecast
    df["forecast_wind"] = df["DE_50hertz_wind_forecast"] + \
                          df["DE_amprion_wind_forecast"] + \
                          df["DE_tennet_wind_forecast"] + \
                          df["DE_transnetbw_wind_forecast"]

    # calculate the total solar forecast
    df["forecast_solar"] = df["DE_50hertz_solar_forecast"] + \
                           df["DE_amprion_solar_forecast"] + \
                           df["DE_tennet_solar_forecast"] + \
```

```
                            df["DE_transnetbw_solar_forecast"]

    # calculate the total forecast
    df["forecast_total"] = df["forecast_wind"] + df["forecast_wind"]

    # a list of the individual TSO's forecast and generation reports ...
    drop_list = ["DE_50hertz_wind_forecast",
                 "DE_amprion_wind_forecast",
                 "DE_tennet_wind_forecast",
                 "DE_transnetbw_wind_forecast",
                 "DE_50hertz_solar_forecast",
                 "DE_amprion_solar_forecast",
                 "DE_tennet_solar_forecast",
                 "DE_transnetbw_solar_forecast",
                 'DE_50hertz_solar_generation',
                 'DE_50hertz_wind_generation',
                 'DE_amprion_solar_generation',
                 'DE_amprion_wind_generation',
                 'DE_amprion_wind_onshore_generation',
                 'DE_tennet_solar_generation',
                 'DE_tennet_wind_generation',
                 'DE_tennet_wind_offshore_generation',
                 'DE_tennet_wind_onshore_generation',
                 'DE_transnetbw_solar_generation',
                 'DE_transnetbw_wind_generation',
                 'DE_transnetbw_wind_onshore_generation']

    # ...is used to drop these columns
    for element in drop_list:
        df.drop(element, axis=1, inplace=True)

    # generate dummy variables for hour, month and weekday
    dummy_list = ['hour',
                  'month',
                  'weekday']
    for element in dummy_list:
        dummy_table = pd.get_dummies(df[element], prefix=element,
prefix_sep="_")
        df = pd.concat([df, dummy_table], axis=1)

    # save the resulting df
    df.to_pickle("00_OPSD_downsized.pickle")
    create_header_overview(df, "00_Headers_downsized.txt")
```

b. Data Manipulation

```
"""
Task:   Preparing of the input dataFrames

        The 'get_dict' function returns a dictionary with all necessary

        This reduces the necessary amount of code at the following points

        of the model.

        There are three different time frames for every topic of interest:

        2010 - 2014:   for classifiers capable of large input data sets

        2014:          for classifiers requiring smaller input data sets

        2015:          for the evaluation

        time series within the dictionary:

                -  "df_10_14" -> complete df with all columns for 2010-2014
                -   "df_15" -> complete df with all columns for 2015
                -   "X": X -> features 2010-2015
                -   "y": y -> labels 2010-2015
                -   "X_14" -> features 2014
                -   "y_14" -> labels 2014
                -   "X_eval" -> features 2015
                -   "y_eval" -> labels 2015
                -   "prices_eval_10_14" -> timeseries of prices 2010-2014
                -   "prices_eval_14" -> timeseries of prices 2014
                -   "prices_eval_15" -> timeseries of prices 2015
                -   "GAMS_result_10_14" -> balance of the LP for 2010-2014
                -   "GAMS_result_14": balance of the LP for 2014
                -   "GAMS_result_15" balance of the LP for 2015



The script performs the following steps
        - adding lag variables
        - importing and labeling the data (linear programming results)
        - replace Nan - values

        - train/test/evaluation - Split
"""
import pandas as pd
from StorageLogic import runLogic


################################################

#               >>  Functions  <<                #

################################################

def get_dict():
    """
    This function performs steps. to be called from other scripts
```

```
    :return: dictionary with all necessary values
    """
    def make_label(val, threshold=0.1):
        # label the results from the LP (Classification)
        if val > threshold:
            return 1
        else:
            return 0
    # load the processed OPSD Dataframe
    dataFrame = pd.read_pickle("00_OPSD_downsized.pickle")

    # a list of elements with non-numeric values
    # including the absolute day variable
    del_list = ["absolute_hour",
                'utc_timestamp',
                'Time',
                'day']

    for element in del_list:
            dataFrame.drop(element, axis=1, inplace=True)

    # add lag variables for load and day ahead price
    for shift in range(1, 25):
        dataFrame["DE_price_day_ahead_T-{}".format(shift)] =
dataFrame["DE_price_day_ahead"].shift(shift)
        dataFrame["DE_load__T-{}".format(shift)] =
dataFrame["DE_load_"].shift(shift)
        # shift for value of last same time last week
    dataFrame["DE_price_day_ahead_T-{}".format("last week")] =
dataFrame["DE_price_day_ahead"].shift(168)
    dataFrame["DE_load__T-{}".format("last_week")] =
dataFrame["DE_load_"].shift(168)

    # make moving average
    rolling_means = [4, 24, 168]
    for RM in rolling_means:
        dataFrame["DE_price_day_ahead_RM-{}".format(RM)] = \
            dataFrame["DE_price_day_ahead"] \
                .rolling(window=RM, min_periods=1).mean()

        dataFrame["DE_load__RM-{}".format(RM)] = \
            dataFrame["DE_load_"]. \
                rolling(window=RM, min_periods=1).mean()

        dataFrame["DE_residual_load__RM-{}".format(RM)] = \
            dataFrame["DE_residual_load"]. \
                rolling(window=RM, min_periods=1).mean()

    # set timeframe
    dataFrame = dataFrame[dataFrame.year >= 2010]
    dataFrame = dataFrame[dataFrame.year <= 2015]

    # shift the forecast columns, so that they are in the correct row
    dataFrame["forecast_wind"] = dataFrame["forecast_wind"].shift(-1)
    dataFrame["forecast_solar"] = dataFrame["forecast_solar"].shift(-1)
    dataFrame["forecast_total"] = dataFrame["forecast_total"].shift(-1)

    # load the results file from the LP and shift it
    GAMS_df = pd.read_csv("GAMS\\results_GAMS_10-15_TOTAL_single_wo_INFO_.csv",
sep=";")
```

```
GAMS_df = GAMS_df[["Feed_in", "Feed_out"]]
GAMS_df = GAMS_df.shift(-1)

# combine features and labels
dataFrame = pd.DataFrame.join(dataFrame.reset_index(), GAMS_df)
dataFrame.drop('cet_cest_timestamp', axis=1, inplace=True)

# transform the label into categorical data
dataFrame["Feed_in"] = list(map(make_label, dataFrame.loc[:, "Feed_in"]))
dataFrame["Feed_out"] = list(map(make_label, dataFrame.loc[:, "Feed_out"]))
# correct setting of labels
dataFrame.loc[:, "behave"] = dataFrame.Feed_in - dataFrame.Feed_out
dataFrame.drop("Feed_in", axis=1, inplace=True)
dataFrame.drop("Feed_out", axis=1, inplace=True)

# replace "nan"  values with last valid value
# replace remaining "nan" with 0
dataFrame.fillna(method="ffill", inplace=True)
dataFrame.fillna(0, inplace=True)

# slice the Data frames into train/test sets and evaluation sets
df_10_14 = dataFrame[dataFrame["year"] <= 2014].copy()
df_14 = dataFrame[dataFrame["year"] == 2014].copy()
df_15 = dataFrame[dataFrame["year"] == 2015].copy()

# make split features and labels for...
# ...optimization (2010- 2014)
X = df_10_14.drop("behave", axis=1)
y = df_10_14["behave"]

# ...optimization (2014)
X_14 = df_14.drop("behave", axis=1)
y_14 = df_14["behave"]

# ..evaluation
X_eval = df_15.drop("behave", axis=1)
y_eval = df_15["behave"]

# shift the prices, to match the features
prices_eval_10_14 = df_10_14["DE_price_day_ahead"].shift(-1)
prices_eval_15 = df_15["DE_price_day_ahead"].shift(-1)
prices_eval_14 = df_14["DE_price_day_ahead"].shift(-1)

# calculate the the profit according the the storage logic
GAMS_result_10_14 = runLogic("GAMS",
                             price_series=prices_eval_10_14,
                             signals=y,
                             signal_format="1").balance
GAMS_result_14 = runLogic("GAMS",
                             price_series=prices_eval_14,
                             signals=y_14,
                             signal_format="1").balance
GAMS_result_15 = runLogic("GAMS",
                          price_series=prices_eval_15,
                          signals=y_eval,
                          signal_format="1").balance

# populate the dictionary to return
return_dict = {"df_10_14": df_10_14,
               "df_15": df_15,
```

```
                    "X": X,
                    "y": y,
                    "X_14": X_14,
                    "y_14": y_14,
                    "X_eval": X_eval,
                    "y_eval": y_eval,
                    "prices_eval_10_14": prices_eval_10_14,
                    "prices_eval_14": prices_eval_14,
                    "prices_eval_15": prices_eval_15,
                    "GAMS_result_10_14":GAMS_result_10_14,
                    "GAMS_result_14":GAMS_result_14,
                    "GAMS_result_15":GAMS_result_15
                    }
    return return_dict
```

c. Evaluation Framework (Storage Logic)

```
"""
Name: StorageLogic
Task: initialize a battery with and providing a function taking a signal and a
price timeseries to simulate a trading process

the battery object allows to simulate a trading process,
while honoring the physical limitations of the EES
simultaneously the objects keeps record of all actions (self.history)
"""

import datetime
import pandas as pd
import numpy as np

################################################
#                >>  Objects  <<                #
################################################


class battery:
    def __init__(self,
                 self_discharge=0.007,
                 volume=1,
                 efficiency=0.825,
                 storage_speed=0.25,
                 balance=0,
                 storage_level=0,
                 signal_modificator=0,
                 initializer="Annon",
                 max_load_cycles = 7000,
                 investment_cost=700000,
                 ):

        self.investment_cost = investment_cost
        self.max_load_cycles = max_load_cycles
        self.minimal_gain_per_load_cycle = investment_cost/max_load_cycles
        self.efficiency = efficiency
        self.signal_modificator = signal_modificator
        self.balance = balance
        self.storage_level = storage_level
        self.volume = volume
        self.storage_speed = storage_speed
        self.self_discharge = self_discharge
        self.initializer = initializer  # Todo implement

        self.history = {  # todo behavouirtracker
            "storagelevel": [],
            "delta_storage_level": [],
            "balance": [],
            "performance": [],
            "price": [],
            "signal": [],
            "activity": []}

################################################
```

```
#                >>  Functions  <<                #

################################################


        def mk_report_dataFrame(self, suffix=False, Path="BAT_History",
filename=False, date=False):
            """
            this function transforms the .self.history to a pandas' DataFrame
            :param suffix: append a name specifying the algorithm.
            :param Path: Path to the output file. default "BAT_History"
            :param filename: if true save as BAT_History_{}.csv
            :param date: adds the date to the file name
            :return: df with history
            """

            if not suffix:
                suffix = self.initializer
            return_df = pd.DataFrame(self.history)
            return_df = return_df.add_suffix(("___"+suffix))
            if filename:
                filename = "{}\\BAT_History_{}".format(Path, filename)
                if date:
                    filename += "_" + str(datetime.date.today())
                filename += ".csv"
                return_df.to_csv(filename, sep=";")

            return return_df

        def store_energy(self, current_price, signal):
            """
            calculate the storing process
            document the storing process
            :param current_price: input price of bat.behave
            :param signal: signal
            :return:
            """
            storable_energy = min(signal, self.volume - self.storage_level)

            # calculate the performance
            performance = storable_energy/self.efficiency * current_price * -1
            self.balance += performance
            # physical storing
            self.storage_level += storable_energy

            # update history
            self.history["performance"].append(performance)
            self.history["balance"].append(self.balance)
            self.history["delta_storage_level"].append(storable_energy)

            # plausibility check
            if self.storage_level > self.volume:
                print("LogicalError")
                quit()

        def sell_energy(self, current_price, signal):
            """
            Calculate the selling process
```

```
        document the selling process
        :param current_price: input price of bat.behave
        :param signal: signal
        :return:
        """
        sellable_energy = min(signal * -1, self.storage_level)

        # calculate the performance
        performance = sellable_energy * current_price
        self.balance += performance

        # physical storing
        self.storage_level -= sellable_energy

        # update history
        self.history["performance"].append(performance)
        self.history["balance"].append(self.balance)
        self.history["delta_storage_level"].append(sellable_energy * -1)

        # plausibility check
        if self.storage_level < 0:
            print("Error1")

    def behave(self, signal, current_price):

        self.history["price"].append(current_price)
        self.history["signal"].append(signal)

        if signal > 0 and self.storage_level < self.volume:
            self.store_energy(current_price, signal)
            self.history["activity"].append("storing..")
            # store

        elif signal < 0 and self.storage_level > 0:
            self.sell_energy(current_price, signal)
            self.history["activity"].append("selling..")
            # sell

        elif signal == 0:
            # wait
            self.history["performance"].append(0)
            self.history["balance"].append(self.balance)
            self.history["delta_storage_level"].append(0)
            self.history["activity"].append("waiting..")
        else:
            # storage cap reached
            self.history["performance"].append(0)
            self.history["balance"].append(self.balance)
            self.history["delta_storage_level"].append(0)
            if self.storage_level == 1:
                self.history["activity"].append("I'm full")
                # "will never happen"

            if self.storage_level == 0:
                self.history["activity"].append("I'm empty")

        if (signal < -1) or (signal > 1):
            # Error
            print("Error, signal is ", signal)
            exit()
```

```
        self.history["storagelevel"].append(self.storage_level)
        ## self-discharge is in %
        self.storage_level *= (1 - self.self_discharge/100)

################################################
#               >>  Functions  <<              #
################################################

def simple_signal(price_series, sell_price=40, buy_price=40, maxStorageSpeed=1):
    """
    This function evaluates the price series and produces simple storage signals
    :param price_series: a times series of prices (pd.Series,list etc)
    :param sell_price: lower threshold for selling energy
    :param buy_price: upperthreshold for buying energy
    :return:list of signals for Storagelogic.py
    """

    signal = []
    for price in price_series:
        if price < buy_price:
            signal.append(maxStorageSpeed)
        elif price > sell_price:
            signal.append(maxStorageSpeed * -1)
        else:
            signal.append(0)
    return signal

def runSimpleLogic(price_series, initializer="SimpleLogic", buy_price=35,
sell_price=38):
    """
    Start a run with a price series
    Compute the signal by itself, depending on given parameters of buy-price and
sell_price
    :param initializer: name of the initializer of the Battery
    :param price_series:
    :param buy_price:
    :param sell_price:
    :return: A Battery, which already performed a cycle run through the
priceseries
    """

    simple = simple_signal(price_series=price_series, buy_price=buy_price,
sell_price=sell_price)
    bat = battery(initializer=initializer)
    for signal, price in zip(simple, price_series):

        bat.behave(signal, price)
    return bat

def runLogic(initializer, price_series, signals, signal_format):
    """
    Run the storage logic with given a price_series and signals
    :param initializer: name of the initializer of the Battery
    :param price_series:
    :param signals:
    :param signal_format:to format the input signal to a consistent value
    :return: A Battery, which already performed a cycle run through the price
series
    """
    bat = battery(initializer=initializer)
```

```
    signals = np.array(signals)
    price_series = np.array(price_series)
    if signal_format == "0.1":
        for signal, price in zip(signals, price_series):
            bat.behave(signal, price)
    if signal_format == "1":
        price_series[-1] = 0
        assert len(signals) == len(price_series)

        for signal, price in zip(signals, price_series):
            bat.behave(signal*bat.storage_speed, price)
    else:
        raise TypeError("Signaltype is not correct!")
    return bat
```

d. Linear Programming Model

```
$ontext
#####################
storage optimization
#####################
 This model solves the optimal storage behavior for X years
$offtext

Option threads=-1;
Option Reslim=10000;

         sets t/t0*t52583/
*         sets t/t1*t24/
;
         parameter        price /
$include C:\Python\Masterarbeit2.0\GAMS\GAMS_import_1d_2010-2015.txt
*$include test_t24.txt
/;

*Variables to define the properties of the storage technology
*        stor_eff                ..the loss during the feed in AND feed out
*                                  process
*        selfdischarge           ..percentual loss due to self discharge
*        max_charge_speed        ..maximum charge speed in dependency of the
*                                  installed capacity

scalar   stor_eff,
         selfdischarge,
         max_feed_speed;
         stor_eff = 0.825;
         selfdischarge = 0.007;
         max_feed_speed = 0.25;

positive variable
         feed_in(t),
         feed_out(t),
         stor_lvl(t);
                 feed_in.up(t)   = max_feed_speed;
                 feed_out.up(t)  = max_feed_speed;
                 stor_lvl.up(t)     = 1

variables
         hourly_profit(t),
         profit_acc(t);

free variable
         balance;

equations
         stor_lvl_eq(t),
*only for results file
         accumulate_profit(t),
         calc_hourly_profit(t),
*objective function
         calculate_balance;

stor_lvl_eq(t)..         stor_lvl(t) =e=
                                 stor_lvl(t-1)
```

```
                                *(1-selfdischarge/100)
                                + feed_in(t)
                                - feed_out(t);

calc_hourly_profit(t)..     hourly_profit(t) =e= feed_out(t)
                                               * price(t)
                                               -(feed_in(t)
                                               * price(t)
                                               /stor_eff);

accumulate_profit(t)..      profit_acc(t)    =e= profit_acc(t-1)
                                               + hourly_profit(t);

calculate_balance..         balance          =e= sum(t,  feed_out(t)
                                                        * price(t))
                                               -sum(t, feed_in(t)
                                                        * price(t)/stor_eff);


model storOpt /all/;
solve storOpt using LP maximize balance;

File output/
"results_GAMS_10-15_TOTAL_single.csv"

/;
File output1/
"results_GAMS_10-15_TOTAL_single_wo_INFO_.csv"
/;

output.nd=5;
output1.nd=5;
PUT output;

Put 'RUNTIME;',system.time";"/;
Put 'RUNDATE;',system.date";"/;
Put 'balance;',balance.l";"/;
Put 'stor_eff;',stor_eff";"/;
Put 'selfdischarge;',selfdischarge";"/;
Put 'max_feed_speed;',max_feed_speed";"/;
Put 'Timesstep;Price;Storage_Level;Feed_in;Feed_out;Balance;'/;

Loop(t,PUT                                                    t.TL";"price(t)";"
stor_lvl.l(t)";"feed_in.l(t)";"feed_out.l(t)";"hourly_profit.l(t)";"profit_acc.l
(t)";"/)
putclose

put output1;
Put "price;Storage_Level;Feed_in;Feed_out;hourly_profit;profit_acc;"/;
Loop(t,PUT
price(t)";"stor_lvl.l(t)";"feed_in.l(t)";"feed_out.l(t)";"hourly_profit.l(t)";"p
rofit_acc.l(t)";"/)
putclose
```

e. Optimization (Grid Search)

The following scripts perform the grid search for the different classifiers, resulting in the optimal set of hyperparameters for the training set (2010-2014/2014) for the respective classifier. The results including the hyperparameters are stored in a csv file

f. K-Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
import pandas as pd

from Datenaufbereitung import get_dict
from StorageLogic import runLogic

output = []
INFOS = get_dict()


# import 2014' Dataframes

X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]


# Scorer calculates the score

f1_score = make_scorer(f1_score, average="weighted")


# loop over scaler-type and number of components for (PCA)
for scaler_ in [StandardScaler, QuantileTransformer]:
    scaler = scaler_()
    X_train_sc = scaler.fit_transform(X_14)
    for components in [20, 40, 50, 100, 119]:
        cv_results_table_name = "KNN\\KNN_{}_{}.xls".format(scaler_.__name__,
components)

        pca = PCA(components)
        X_train_pca = pca.fit_transform(X_train_sc)
        params = {"n_neighbors": [10, 11, 13, 16, 20],
                  "n_jobs": [-1],
                  "weights": ["uniform", "distance"],
                  "p": [1, 2]}

        clf = GridSearchCV(KNeighborsClassifier(), params, cv=5,
scoring="f1_weighted", verbose=1)
        clf.fit(X_train_pca, y_14)

        # populate output Datatframe
        cv_result_df = pd.DataFrame(clf.cv_results_)
        cv_result_df = cv_result_df[['mean_test_score',
                                     'std_test_score',
                                     'mean_train_score',
                                     'std_train_score',
```

```
                                'param_n_neighbors',
                                'param_p',
                                'param_weights',
                                'rank_test_score', ]]
# save output
cv_result_df = cv_result_df.sort_values("rank_test_score")
cv_result_df.to_excel(cv_results_table_name)
```

g. Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
import pandas as pd
from Datenaufbereitung import get_dict

# import DataFrames
INFOS = get_dict()
X = INFOS["X"]
y = INFOS["y"]

# Scorer calculates the score

f1_score = make_scorer(f1_score, average="weighted")

# set up classifier
pipe = make_pipeline(DecisionTreeClassifier())
param_grid = {'decisiontreeclassifier__min_samples_leaf':[1,20,40,60,100],
              'decisiontreeclassifier__min_samples_split':[2,50,100,200],
              'decisiontreeclassifier__max_depth':[100,50,25,10],
              'decisiontreeclassifier__criterion':["gini", "entropy"],
              'decisiontreeclassifier__class_weight':["balanced"]
             }

# set up gridsearch parameter
gs = GridSearchCV(pipe,
                  param_grid=param_grid,
                  cv=5,
                  n_jobs=-1,
                 verbose=3)

# fit classifier
gs.fit(X_train, y_train)

# save result
pd.DataFrame(gs.cv_results_).to_excel("DT\\DecisionTree.xls")
```

h. Random Forest

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
import pandas as pd
from Datenaufbereitung import get_dict

# import input DataFrames
INFOS = get_dict()
X = INFOS["X"]
y = INFOS["y"]

# scorer calculates the score

f1_score = make_scorer(f1_score, average="weighted")

# set up classifier
pipe = make_pipeline(RandomForestClassifier())
param_grid = {'randomforestclassifier__n_estimators':[10,20,50,100],
              'randomforestclassifier__max_features':[10,30,60,90,"auto"],
              'randomforestclassifier__criterion':["gini", "entropy"],
              'randomforestclassifier__class_weight':["balanced"]

             }
# set up grid search
gs = GridSearchCV(pipe,
                  param_grid=param_grid,
                  cv=5,
                  n_jobs=-1,
                 verbose=3)

# fit classifier
gs.fit(X, y)

# save result
pd.DataFrame(gs.cv_results_).to_excel("RF\\RandomForest.xls")
```

i. Logistic Regression

For the logistic regression two versions were examined: one with a PCA and one without a PCA.

i. Grid Search without PCA

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
import pandas as pd
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from Datenaufbereitung import get_dict

# import DataFrames
INFOS = get_dict()
X = INFOS["X"]
y = INFOS["y"]

# scorer calculates the score
f1_score = make_scorer(f1_score, average="weighted")

# loop over different scalers
for scaler in [StandardScaler, QuantileTransformer]:

    # scale the input data
    scaler = scaler()
    X_train = scaler.fit_transform(X)

    # set up classifier
    pipe = make_pipeline(LogisticRegression())
    param_grid = {'logisticregression__penalty':["l1","l2"],
              'logisticregression__C':[0.0001,0.001,0.01,0.1,1,10,100],
              'logisticregression__solver':["saga"],
              'logisticregression__class_weight':["balanced"],
              'logisticregression__max_iter':[1000]
             }

    # set up grid search
    gs = GridSearchCV(pipe,
                      param_grid=param_grid,
                      cv=5,
                      n_jobs=-1,
                     verbose=3,
                     scoring = f1_score)
    # fit classifiers
    gs.fit(X_train, y)

    # save output

pd.DataFrame(gs.cv_results_).to_excel("JuPyter\\LogReg\\ohne_PCA_{}.xls".format(
scaler.__name__))
```

ii. With Polynomial Feature Selection and PCA

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.decomposition import PCA
import pandas as pd
from Datenaufbereitung import get_dict


# import DataFrames

INFOS = get_dict()
X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

# scorer calculates the score
f1_score = make_scorer(f1_score, average="weighted")

# loop over scalers and PCA components
for scaler_ in [StandardScaler, QuantileTransformer]:
    for com in [10,100,500,1000]:

        # set up scaler and scale the features
        scaler_name = scaler_.__name__
        scaler = scaler()
        X_train = scaler.fit_transform(X_14)

        # set up polinomalizer
        poly = PolynomialFeatures()
        X_train = poly.fit_transform(X_train)

        # set up PCA
        pca = PCA(com)
        X_train = pca.fit_transform(X_train)

        # set up classifier
        pipe = make_pipeline(LogisticRegression())
        param_grid = {'logisticregression__penalty':["l2"],
                      'logisticregression__C':[0.01,0.1,1,10,100],
                      'logisticregression__solver':["saga"],
                      'logisticregression__class_weight':["balanced"],
                      'logisticregression__max_iter':[50000]
                     }
        # set up grid search
        gs = GridSearchCV(pipe,
                          param_grid=param_grid,
                          cv=5,
                          n_jobs=-1,
                          verbose=3,
                          scoring = "f1_weighted")

        # train classifier
        gs.fit(X_train, y_14)

        # save output
```

```
pd.DataFrame(gs.cv_results_).to_excel("LogReg\\Poly_{}_PCA_{}.xls".format(com,
scaler_name))
```

j. Support Vector Classifier

i. Without Polynomial Feature Enrichment

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.decomposition import PCA

from sklearn.metrics import make_scorer
import pandas as pd

from Datenaufbereitung import get_dict

# import DataFrames
INFOS = get_dict()
X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

# loop over the scalers
for scaler_ in [StandardScaler, QuantileTransformer]:
    for components in [10,20,30,40,50,60,70,80,90,100,119]:

        # scores calculates the score
        f1_score = make_scorer(f1_score, average="weighted")

        scaler = scaler_()
        X_train = scaler.fit_transform(X_14)

        pca = PCA(components)
        X_train = scaler.fit_transform(X_train)

        cs = [0.01,0.1,1,10,100,1000,10000,100000]
        gammas = [0.00001,0.0001,0.001,0.01]
        tuned_parameters = [{'kernel': ['rbf'], 'gamma': gammas,'C': cs},
                            {'kernel': ['linear'], 'C': cs}]

        gs = GridSearchCV(SVC(max_iter=100000),
                          param_grid=tuned_parameters,
                          cv=5,
                          n_jobs=-1,
                          verbose=3,
                          scoring = f1_score)
        gs.fit(X_train, y_14)

        pd.DataFrame(gs.cv_results_).sort_values("rank_test_score").\
            to_excel("JuPyter\\SVC\\{}_PCA_{}.xls".format(components,
            scaler.__name__))
```

ii. With Polinomial Feature Selection and PCA

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.decomposition import PCA

import pandas as pd
from Datenaufbereitung import get_dict()

# Import Dataframe
INFOS = get_dict()
X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

# loop over scalers
for scaler_ in [StandardScaler, QuantileTransformer]:
    # loop over comonents for PCA
    for components in [10,100,500,1000,2000]:
        # calculates the score
        f1_score = make_scorer(f1_score, average="weighted")

        # scale the features
        scaler_name = scaler_.__name__
        scaler = scaler_()
        X_train = scaler.fit_transform(X_14)

        # polinomial enrichment
        poly = PolynomialFeatures()
        X_train = poly.fit_transform(X_train)

        # decomposition
        pca = PCA(components)
        X_train = pca.fit_transform(X_train)
        X_test = pca.transform(X_test)

        cs = [0.01,0.1,1,10,100,1000,10000,100000]
        gammas = [0.00001,0.0001,0.001,0.01]
        tuned_parameters = [{'kernel': ['rbf'], 'gamma': gammas,'C': cs},
                            {'kernel': ['linear'], 'C': cs}]

        gs = GridSearchCV(SVC(max_iter = 2000),
                          param_grid=tuned_parameters,
                          cv=5,
                          n_jobs=-1,
                          verbose=3,
                          scoring = "f1_weighted")
        gs.fit(X_train, y_14)

        pd.DataFrame(gs.cv_results_).sort_values("rank_test_score").\
            to_excel("JuPyter\\SVC\\Poly_{}_PCA_{}.xls".format(components,
scaler_name))
```

k. Neuronal Network

```
import os
# reduce log entries
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop, SGD
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.preprocessing import QuantileTransformer
from sklearn.utils import class_weight
import numpy as np

from Datenaufbereitung import get_dict
INFOS = get_dict()

def create_and_train(INFOS):
    """ callable function of the Neuronal Net"""
    X = INFOS["X"]
    y = INFOS["y"]

    # for evaluation
    X_15 =  INFOS["X_15"]
    X_14 =  INFOS["X_14"]

    # reformat the label into a vector with three dimensions
    y = keras.utils.to_categorical(y, num_classes=3)

    scaler = QuantileTransformer()
    X = scaler.fit_transform(X)
    X_14 = scaler.fit_transform(X_14)
    X_15 = scaler.fit_transform(X_15)

    # compute the weight of the classes)
    classweight = class_weight.compute_class_weight('balanced',
np.unique(INFOS["y"]),INFOS["y"])

    # Model Parameter
    my_optimizer = RMSprop(lr=0.01)
    loss_func = "categorical_crossentropy"

    input_layer = "sigmoid"
    output_layer = "softmax"
    activation_func = "hard_sigmoid"
    nr_hidden_layer = 2

    layer_size = X.shape[1]
    nr_hidden_layer_nodes = X.shape[1]

    epochs = 100
    batch_size = 1000

    ## Callbacks
    # methodes to influence the learning process
    # stop if loss function dose not improve 4 times in a row
    early_stopping_mentor = EarlyStopping(monitor="categorical_crossentropy",
patience=4, mode="auto")
```

```
    # reduce lr if loss function stagnates
    reduce_lr = ReduceLROnPlateau(monitor='categorical_crossentropy',
factor=0.5,
                                  patience=1, min_lr=0.0001)

    # set up model and add layers
    model = Sequential()

    # add inupt layer
    model.add(Dense(nr_hidden_layer_nodes, input_dim=layer_size,
activation=input_layer))

    for _ in range(nr_hidden_layer):
        # add an arbitrary number of layers
        model.add(Dense(int(nr_hidden_layer_nodes),
activation=activation_func))
        # add a Dropout layer that mutes random x percents of the nodes
        model.add(Dropout(0.1))

    # add specific outputlayer
    model.add(Dense(3, activation=output_layer))
    model.compile(loss=loss_func, optimizer=my_optimizer,
                  metrics=["categorical_crossentropy", "categorical_accuracy"])

    ## Fit the model
    model.fit(X,
              y,
              callbacks=[early_stopping_mentor, reduce_lr],
              epochs=epochs,
              batch_size=batch_size,
              verbose=1,
              class_weight=classweight,
          )

    def mk_output(output):
        """ construction of the return dictionary"""
        signal = []
        for step in output:
            step = list(step)
            one_hot = max(step)

            if step.index(one_hot) == 0:
                signal.append(0)
            elif step.index(one_hot) == 1:
                signal.append(1)
            elif step.index(one_hot) == 2:
                signal.append(-1)
            else:
                print("error", one_hot, np.where(output == one_hot))
        return signal
    predictions = {"X" : mk_output(model.predict(X)),
                   "X_14": mk_output(model.predict(X_14)),
                   "X_15": mk_output(model.predict(X_15))}


    return predictions
```

I. Training and Evaluation

```
"""
This script performs the actual training of the classifiers based on the data of
2010-2014 respectively only 2014 and the optimal hyperparameters found during
the optimization. Then the classifiers predict the optimal signals for the 2015.
The resulting signals are used to evaluate the economic performance of the
classifier.

Additionally the alternative strategies are evaluated.
"""



import datetime
import Datenaufbereitung
import pickle
import numpy as np

import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score,accuracy_score
from StorageLogic import runLogic, simple_signal



def calculate_load_cyles(bat):
    """
    This function estimates the load cycles by adding all the storage input
together """
    a_ray = np.array(bat.history["delta_storage_level"])
    feed_ins = a_ray[a_ray > 0]

    return feed_ins.sum()


def heatmap(values, xlabel, ylabel, xticklabels, yticklabels, cmap=None,
            vmin=None, vmax=None, ax=None, fmt="%0.2f"):
    """
    returns the a fig used to generate the confusion matrix
    """
    if ax is None:
        ax = plt.gca()
    font = {'family': 'monospace',
            'size': 15}
    import matplotlib
    matplotlib.rc('font', **font)
    # plot the mean cross-validation scores
    img = ax.pcolor(values, cmap=cmap, vmin=vmin, vmax=vmax)
    img.update_scalarmappable()
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_xticks(np.arange(len(xticklabels)) + .5)
    ax.set_yticks(np.arange(len(yticklabels)) + .5)
    ax.set_xticklabels(xticklabels, rotation=0)
```

```
    ax.set_yticklabels(yticklabels)
    ax.set_aspect(1)

    for p, color, value in zip(img.get_paths(), img.get_facecolors(),
                               img.get_array()):
        x, y = p.vertices[:-2, :].mean(0)
        if np.mean(color[:3]) > 0.5:
            c = 'k'
        else:
            c = 'w'
        ax.text(x, y, fmt % value, color=c, ha="center", va="center")

    return img

################################################
#              >>  OUTPUT SET UP  <<           #
################################################

with open("summary_table_{}.csv".format(datetime.date.today()), "w") as file:
    """
    Create a table/header for the output file
    """
    columns = ["name",
               "Accuracy (Train)",
               "Accuracy (Test)",
               "F1 (Train)",
               "F1 (Test)",
               "profit 2014",
               "rel profit 2014",
               "profit 2015",
               "rel profit 2015",
               "load cycles",
               "profit per load cycle",
               "LCOE"]
    for column in columns:
        file.write(column)
        file.write(";")
    file.write("\n")


def write_infos_in_table(name, clf, Mode=None):
    """
    writes the results into the table
    :param name: Name of the classifier
    :param clf: classifier
    :param Mode: different calculations depending on the caller of ther function
    """
    if Mode == "NN":
        pred_14 = clf["X_14"]
        pred_15 = clf["X_15"]

    elif Mode == "back2back":
        pred_14 = list(y_14)[clf:] + list(y_14)[:clf]
        pred_15 = list(y_15)[clf:] + list(y_15)[:clf]

    elif Mode == "simple":
        pred_14 = simple_signal(prices_14)
        pred_15 = simple_signal(prices_15)

    elif Mode == "GAMS":
```

```
        pred_14 = y_14
        pred_15 = y_15

    else:
        pred_14 = clf.predict(X_14)
        pred_15 = clf.predict(X_15)

    # calculate the performance for the year 2014 / train
    report_14 = runLogic(name + "_14",
                         price_series=prices_14,
                         signals=pred_14,
                         signal_format="1")

    # calculate the performance for the year 2015 / test
    report_15 = runLogic(name + "_15",
                         price_series=prices_15,
                         signals=pred_15,
                         signal_format="1")

    lc = calculate_load_cyles(report_15)

    # calculate metrics
    name = name
    acc_train = accuracy_score(pred_14, y_14)
    acc_test = accuracy_score(pred_15, y_15)
    f1_train = f1_score(pred_14, y_14, average="weighted")
    f1_test = f1_score(pred_15, y_15, average="weighted")
    profit_14 = report_14.balance
    rel_profit_14 = report_14.balance / GAMS_14
    profit_15 = report_15.balance
    rel_profit_15 = report_15.balance / GAMS_15
    load_cycles_15 = lc
    profit_per_loadcycle = profit_15 / lc

    report_15.mk_report_dataFrame(suffix=name, filename=name, date=True)
    output = [name,
              acc_train,
              acc_test,
              f1_train,
              f1_test,
              profit_14,
              rel_profit_14,
              profit_15,
              rel_profit_15,
              load_cycles_15,
              profit_per_loadcycle]

    # save results
    with open("summary_table_{}.csv".format(datetime.date.today()), "a") as
file:
        for cell in output:
            file.write(str(cell))
            file.write(";")
        file.write("\n")

    scores_image = heatmap(confusion_matrix(y_15, pred_15),
                           xlabel='Predicted label',
                           ylabel='True label',
                           xticklabels=["discharge", "wait", "charge"],
                           yticklabels=["discharge", "wait", "charge"],
```

```
                            cmap=plt.cm.gray_r, fmt="%d")

    plt.title("{}\n{}".format(name.replace("_","").title(),f1_test))
    plt.gca().invert_yaxis()
    plt.tight_layout()
    # plt.subplots_adjust(left=0.0, right=0.75, top=0.92, bottom=0.09)
    # plt.show()
    plt.savefig("pictures\\Confusion Martix\\{}".format(name))
    plt.clf()


##################################################
#                  >>  IMPORT  <<                #
##################################################


INFOS = Datenaufbereitung.get_dict()
pickle.dump(INFOS, open("INFOS.p", "wb"))

X = INFOS["X"]
y = INFOS["y"]

X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

X_15 = INFOS["X_eval"]
y_15 = INFOS["y_eval"]

GAMS_14 = INFOS["GAMS_result_14"]
GAMS_15 = INFOS["GAMS_result_15"]

prices_14 = INFOS["prices_eval_14"]
prices_15 = INFOS["prices_eval_15"]

##################################################
#             >>  CLASSIFIER SETUP  <<           #
##################################################

print(">>> KNN <<<")
from sklearn.neighbors import KNeighborsClassifier

pipe = make_pipeline(QuantileTransformer(),
                     PCA(100),
                     KNeighborsClassifier(n_neighbors=11,
                                          weights="distance",
                                          p=1,
                                          n_jobs=-1
                                          ))

pipe.fit(X_14, y_14)
write_infos_in_table("KNN", pipe)

print(">>> DT <<<")
from sklearn.tree import DecisionTreeClassifier

pipe = make_pipeline(DecisionTreeClassifier(class_weight="balanced",
                                            criterion="entropy",
                                            max_depth=50,
                                            min_samples_leaf=1,
                                            min_samples_split=50))
```

```
pipe.fit(X, y)
write_infos_in_table("Decision Tree", pipe)


print(">>> Random Forest <<<")
from sklearn.ensemble import RandomForestClassifier
pipe = make_pipeline(RandomForestClassifier(class_weight="balanced",
                                            n_estimators=1000,
                                            criterion="entropy",
                                            # max_features=90,
                                            n_jobs=-1))

pipe.fit(X, y)
write_infos_in_table("Random Forest", pipe)


print(">>> LogReg <<<")
from sklearn.linear_model import LogisticRegression

pipe = make_pipeline(QuantileTransformer(),
                     PolynomialFeatures(2),
                     PCA(1000),
                     LogisticRegression(class_weight="balanced",
                                        solver="saga",
                                        penalty="L2",
                                        C=0.1,
                                        max_iter=100000))

pipe.fit(X_14, y_14)
print("LOG still 14")
write_infos_in_table("Logistic Regression", pipe)

print(">>> SVM - RBF <<<")
from sklearn.svm import SVC

pipe = make_pipeline(StandardScaler(),
                     SVC(class_weight="balanced",
                         C=10,
                         gamma = 0.0001,
                         kernel= "rbf",
                         max_iter=100000))

pipe.fit(X_14, y_14)
write_infos_in_table("SVC - RBF", pipe)

print(">>> SVM - Linear <<<")
from sklearn.svm import SVC

pipe = make_pipeline(QuantileTransformer(),
                     SVC(kernel="linear",
                         C = 10,
                         max_iter=1000000))

pipe.fit(X_14, y_14)
write_infos_in_table("SVC - Linear", pipe)

print(">>> NN <<<")
import FF_simple
pred = FF_simple.create_and_train(INFOS)
```

```
write_infos_in_table("Neuronal_Network", clf = pred, Mode="NN")

################################################
#              >>  TRAIN DUMMIES  <<             #
################################################

pred = -168
write_infos_in_table("Shift Week", clf =pred , Mode="back2back")
pred = -24
write_infos_in_table("Shift Day", clf =pred, Mode="back2back")
write_infos_in_table("Simple_signal", clf=pred, Mode="simple")
write_infos_in_table("GAMS", clf=None, Mode="GAMS")
```

m. Additional Calculations

i. Visualization of Correlation between f1-Score and Profit

```
"""
arbitrary model to visualize the correlation between profit and f1-score
"""
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score
from StorageLogic import runLogic
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
import numpy as np
import Datenaufbereitung

# import DataFrames
INFOS = Datenaufbereitung.get_dict()

X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

X_15 = INFOS["X_eval"]
y_15 = INFOS["y_eval"]

GAMS_14 = INFOS["GAMS_result_14"]
GAMS_15 = INFOS["GAMS_result_15"]

prices_14 = INFOS["prices_eval_14"]
prices_15 = INFOS["prices_eval_15"]

################################################
#            >>  CLASSIFIER SETUP  <<          #
################################################

print(">>> LogReg <<<")
cs = np.logspace(-5,6,12)
f1_evals = []
f1_trains = []
profits = []

# loop over differnt cs values in order to obtain different results profits and
f1-scores
for c in cs:
    # Train/Test - split
    X_train, X_test, y_train, y_test = train_test_split(X_14,y_14,test_size=0.2)

    # make a pipeline out of scaler feature selection and logistic regression
    pipe = make_pipeline(StandardScaler(),
                         SelectKBest(k = 25),
                        LogisticRegression(class_weight="balanced",
                                           solver="saga",
                                           penalty="L2",
                                           C=c,
                                           max_iter=10000))
    pipe.fit(X_train, y_train)
```

```
    # calculate metrics with fitted time series
    pred_eval = pipe.predict(X_15)
    f1_eval = f1_score(y_true=y_15, y_pred=pred_eval, average="weighted")
    profit = runLogic(initializer="LR",
                      price_series=prices_15,
                      signals=pred_eval,
                      signal_format="1").balance/GAMS_15

    f1_evals.append(f1_eval)
    profits.append(profit)

# plot the results
plt.plot(cs, f1_evals, label="F1 - Score", color="#428bca")
plt.plot(cs, profits, label="Profits", color="#d9534f")

# decorate the plots
plt.title("Correlation Between F1 - Score and Profit")
plt.ylabel("F1 - Score / Relative Performance")
plt.xlabel("Cs")
plt.ylim([0.5, 1])
plt.xscale("log")
plt.legend()
plt.grid()
plt.show()
```

ii. Effect of Forecast Horizon

```
"""
Testing the effects of the time horizon on the quality of predictions
"""



import pandas as pd
import scipy.stats
import matplotlib.pyplot as plt
plt.style.use("seaborn-whitegrid")
import glob

# import all History tables
df1 = pd.read_csv("C:\Python\Masterarbeit2.2\BAT_History\BAT_History_GAMS_2017-
12-10.csv",sep=";")
history_liste = glob.glob("C:\Python\Masterarbeit2.2\BAT_History\*2017-12-
10.csv")

# set up DataFrame to store all time series
ensamble = pd.DataFrame()

# prepare output header
print("{:20}\t{:>10}\t{:>10}\t{:>10}\t{:>10}".format("Name", "Intercept",
"Slope", "p_value", "R^2"))

# loop over all histories
for df in history_liste:
    # extract the name of the history of by the name of the file
    name = df.split("_")[-2]

    # exception handling
    # only considering well train MLA
    if name in ["signal", "GAMS", "Shift Day", "Shift Week"]:
        continue
    if name == "Network":
        name = "Neuronal_Network"
    if "SVC" in name:
        continue
    # load history and combine it with LP results (GAMS)
    df2 = pd.read_csv(df, sep=";")
    df3 = pd.DataFrame({"GAMS": df1["signal___GAMS"], "NN":
df2["signal___"+name]})

    # # calculate squared error between profit for predicted action and optimal
action
    # df3.loc[:, "diff"] = (df3.NN - df3.GAMS)**2
    # df3["mean"] = df3["diff"].rolling(window = 720, center=True).sum()

    # calculate the moving average of the accuracy
    df3["acc"] = df3.NN == df3.GAMS
    df3["roll acc"] = df3["acc"].rolling(window = 720, center=True).mean()
    plt.plot(df3["roll acc"],label=name)

    y = df3["roll acc"].dropna()
    y = y.values
```

```
    # performe a linear regression on the accuracy
    slope, intercept, r_value, p_value, std_err =
scipy.stats.linregress(range(len(y)), y)
    # print("slope", slope, "intercept", intercept, "r", r_value, "p", p_value,
"std_err", std_err)
    print("{:20}\t{:10.4f}\t{:10.4f}\t{:10.4f}\t{:10.4f}".format(name,
intercept, slope, p_value, r_value**2))

plt.ylim([0,1])

# decorate the plot
plt.title("Temporal Effects\n2015")
plt.ylabel("Accuracy")
plt.xlabel("hours")
plt.legend()
plt.show()
```

iii. Improving the Model's Quality

```
"""
Test the effects of different decision boundaries on the economic performance of
the model
"""
import Datenaufbereitung
import matplotlib.pyplot as plt
from sklearn.preprocessing import  QuantileTransformer
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score,accuracy_score,precision_score,recall_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
from StorageLogic import runLogic

def map(x):
    if x == 2:
        return 1
    elif x == 1:
        return 0
    elif x == 0:
        return -1

INFOS = Datenaufbereitung.get_dict()

X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

X_15 = INFOS["X_eval"]
y_15 = INFOS["y_eval"]

GAMS_14 = INFOS["GAMS_result_14"]
GAMS_15 = INFOS["GAMS_result_15"]

prices_14 = INFOS["prices_eval_14"]
prices_15 = INFOS["prices_eval_15"]

################################################
#             >>  CLASSIFIER SETUP  <<         #
################################################


pipe = make_pipeline(QuantileTransformer(),
                     PCA(100),
                     KNeighborsClassifier(n_neighbors=11,
                                          weights="distance",
                                          p=1,
                                          n_jobs=-1
                                          ))

pipe.fit(X_14, y_14)

proba15 = pipe.predict_proba(X_15)
proba14 = pipe.predict_proba(X_14)

profits_15 = []
profits_14 = []




# loop over different threshold levels
for threshold in range(0, 101):
    threshold /= 100
    signal15 = []
    signal14 = []

    z = proba15.copy()
    for x in z:
    # classify based on threshold
        if x[0] < threshold and x[2] < threshold:
            x[1] = 1
```

```
        else :
            if x[0] > x [2]:
                x[0] = 1
            else:
                x[2] = 1
        signal15.append(map(np.argmax(x)))
    # calculate new profit for 2015 / test
    profit15 = runLogic(initializer="KNN",
                        price_series=prices_15,
                        signals=signal15,
                        signal_format="1").balance / GAMS_15

    z = proba14.copy()
    for x in z:
        # classify based on threshold
        if x[0] < threshold and x[2] < threshold:
            x[1] = 1
        else:
            if x[0] > x[2]:
                x[0] = 1
            else:
                x[2] = 1
        signal14.append(map(np.argmax(x)))

    # calculate profit for 2014 / train
    profit14 = runLogic(initializer="KNN",
                            price_series=prices_14,
                            signals=signal14,
                            signal_format="1").balance / GAMS_14

    profits_15.append(profit15)
    profits_14.append(profit14)

# plot and decorate
plt.plot(profits_14, color= "#d9534f", label="2014 (Train)")
plt.plot(profits_15, color= "#428bca", label="2015 (Test)")

# marke the maiximum and default values for every year
plt.scatter(np.argmax(profits_15),max(profits_15),color= "#428bca", label="max
2015_({})".format(np.argmax(profits_15)/100))
plt.scatter(np.argmax(profits_14),max(profits_14),color= "#d9534f", label="max
2014_({})".format(np.argmax(profits_14)/100) )
plt.scatter(50,profits_15[50],color= "k", label="default 2015_(0.5)" )
plt.xlabel("threshold")
plt.title("Decision Boundaries")
plt.yticks(np.arange(0,1,0.05))
plt.xticks(np.arange(0,100,10))
plt.grid()
plt.ylabel("profit [EUR/a]")

plt.legend()
plt.show()
```

iv. Source Code Model-Based Feature Selection

```
# model based feature selection (normal features)
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=1000,
                                random_state = 42,
                                n_jobs=-1)
forest.fit(X_train, y_train)

importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
feat_labels = X.columns

top10 = indices[:10]
top10_names = feat_labels[top10]


top10_names = [x.replace("DE", "").replace("_"," ").title() for x in
top10_names]
plt.grid("off")
plt.barh(range(1,11),
        sorted(importances[top10]),
        color=blue,
        align='center')
plt.yticks(range(1,11),
        reversed(top10_names),)
plt.xticks([0.01,0.02,0.03,0.04,0.05])

plt.ylim([0, 11])
plt.show()


sorted_importences = sorted(list(importances),reverse=True)

fig, ax1 = plt.subplots()

ax1.plot(sorted_importences, label = "importance of feature", color = blue)
ax1.set_ylabel("single")
ax1.set_yticks([0.01,0.02,0.03,0.04,0.05])
ax2 = ax1.twinx()
ax2.set_ylabel("cumulative")
ax2.plot(np.array(sorted_importences).cumsum(), label = "cumulative", color =
red)
ax2.set_yticks([0.2,0.4,0.6,0.8,1])
ax2.set_ylim([0,1])
# fig.suptitle('Importance of Features\n Random Forrest n=1000')

ax1.set_xlabel('Features')


ax1.grid()
plt.show()


# model based feature selection (normal features)
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=100,
```

```
                                    random_state=42,
                                    n_jobs=-1)
forest.fit(X_trans, y_train)
importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]

feat_labels = [(poly_name_dict[_]) for _ in ploy2.get_feature_names()]
top10 = indices[:10]
top10_names = [feat_labels[_] for _ in top10]

importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]

feat_labels = [(poly_name_dict[_]) for _ in ploy2.get_feature_names()]
top10 = indices[:10]
top10_names = [feat_labels[_] for _ in top10]

top10_names = [x.replace("DE", "").replace("_"," ").replace("*"," *").title()
for x in top10_names]

top10_names = reversed(top10_names)

plt.barh(range(1,11),
        sorted(importances[top10]),
        color=blue,
        align='center')
plt.yticks(range(1,11),top10_names)
plt.ylim([0, 11])

plt.show()

sorted_importences = sorted(list(importances),reverse=True)

fig, ax1 = plt.subplots()

ax1.plot(sorted_importences, label = "importance of feature", color = blue)
ax1.set_ylabel("single")
ax1.set_yticks([0,0.0001,0.0002,0.0003,0.0004,0.0005])
ax2 = ax1.twinx()
ax2.set_ylabel("cumulative")
ax2.plot(np.array(sorted_importences).cumsum(), label = "cumulative", color =
red)
ax2.set_yticks([0,0.2,0.4,0.6,0.8,1])
ax2.set_ylim([0,1])
ax1.set_ylim([0,0.0005])
# fig.suptitle('Importance of Features\n Random Forrest n=1000')

ax1.set_xlabel('Features')


ax1.grid()
plt.show()
```

v. Source Code Performance improvement via Decision Boundaries

```
import Datenaufbereitung
import pickle
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score,accuracy_score,precision_score,recall_score
import numpy as np
from StorageLogic import runLogic, simple_signal


def map(x):
    if x == 2:
        return 1
    elif x == 1:
        return 0
    elif x == 0:
        return -1

INFOS = Datenaufbereitung.get_dict()


X = INFOS["X"]
y = INFOS["y"]

X_14 = INFOS["X_14"]
y_14 = INFOS["y_14"]

X_15 = INFOS["X_eval"]
y_15 = INFOS["y_eval"]

GAMS_14 = INFOS["GAMS_result_14"]
GAMS_15 = INFOS["GAMS_result_15"]

prices_14 = INFOS["prices_eval_14"]
prices_15 = INFOS["prices_eval_15"]

print(">>> KNN _dec <<<")
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest
import numpy as np



X_train, X_test, y_train, y_test = train_test_split(X_14,y_14,test_size=0.2)
from sklearn.svm import SVC


pipe = make_pipeline(QuantileTransformer(),
                     PCA(100),
                     KNeighborsClassifier(n_neighbors=11,
                                          weights="distance",
```

```
                                            p=1,
                                            n_jobs=-1
                                            ))

pipe.fit(X_train, y_train)


pred_test = pipe.predict(X_test)
pred_eval = pipe.predict(X_15)
pred_train = pipe.predict(X_train)

profit = runLogic(initializer="KNN",
                  price_series=prices_15,
                  signals=pred_eval,
                  signal_format="1").balance/GAMS_15



proba15 = pipe.predict_proba(X_15)
proba14 = pipe.predict_proba(X_14)

accuracy_15 = []
accuracy_14 = []
f1score_15 = []
f1score_14 =[]
profits_15 = []
profits_14 = []
precision = []
recall = []

for threshold in range (0, 101):
    threshold /= 100
    signal15 = []
    signal14 = []

    z = proba15.copy()
    for x in z:
        # apply the threshold
        if x[0] < threshold and x[2] < threshold:
            x[1] = 1
        else :
            if x[0] > x [2]:
                x[0] = 1
            else:
                x[2] = 1
        signal15.append(map(np.argmax(x)))


    score_f1_15 = f1_score(signal15,y_15, average="weighted")
    score_acc_15 = accuracy_score(signal15,y_15)
    profit15 = runLogic(initializer="KNN",
                        price_series=prices_15,
                        signals=signal15,
                        signal_format="1").balance / GAMS_15


    z = proba14.copy()
    for x in z:

        if x[0] < threshold and x[2] < threshold:
```

```
            x[1] = 1
        else:
            if x[0] > x[2]:
                x[0] = 1
            else:
                x[2] = 1
        signal14.append(map(np.argmax(x)))

    score_f1_14 = f1_score(signal14,y_14, average="weighted")
    score_acc_14 = accuracy_score(signal14,y_14)

    profit14 = runLogic(initializer="KNN",
                            price_series=prices_14,
                            signals=signal14,
                            signal_format="1").balance / GAMS_14

    print("{} ; profit 15 {:10.4f}; profit 14 {:10.4f}; acc: {:10.4f} ;f1:
{:10.4f}".format(threshold, profit15, profit14, score_acc_15, score_f1_15))

    accuracy_15.append(score_acc_15)
    accuracy_14.append(score_acc_14)
    f1score_15.append(score_f1_15)
    f1score_14.append(score_acc_14)
    profits_15.append(profit15)
    profits_14.append(profit14)
    precision.append(precision_score(signal15,y_15, average="weighted"))
    recall.append(recall_score(signal15,y_15, average="weighted"))



print("max acc train", np.argmax(accuracy_14))
print("max acc test", np.argmax(accuracy_15))
print("max f1 train", np.argmax(f1score_14))
print("max f1 test", np.argmax(accuracy_15))
print("max profit train", np.argmax(profits_14), max(profits_14))
best_thresh = np.argmax(profits_14)/100
print("max profit test", np.argmax(profits_15),max(profits_15))

plt.plot(profits_14, color= "#d9534f", label="2014 (Train)")
plt.plot(profits_15, color= "#428bca", label="2015 (Test)")
plt.scatter(np.argmax(profits_15),max(profits_15),color= "#428bca", label="max
2015_({})".format(np.argmax(profits_15)/100))
plt.scatter(np.argmax(profits_14),max(profits_14),color= "#d9534f", label="max
2014_({})".format(np.argmax(profits_14)/100) )
plt.scatter(50,profits_15[50],color= "k", label="default 2015_(0.5)" )
plt.xlabel("threshold")
plt.title("Decision Boundaries")
plt.yticks(np.arange(0,1,0.05))
plt.xticks(np.arange(0,100,10))
plt.grid()
plt.ylabel("profit [EUR/a]")

plt.legend()
plt.show()
pipe.fit(X_14, y_14)
signal=[]
for x in pipe.predict_proba(X_15):
    if x[0] < best_thresh and x[2] < best_thresh :
        x[1] = 1
```

```
    else:
        if x[0] > x[2]:
            x[0] = 1
        else:
            x[2] = 1
    signal.append(map(np.argmax(x)))

profit = runLogic(initializer="KNN",
                  price_series=prices_15,
                  signals=signal,
                  signal_format="1").balance / GAMS_15

print("XX:",profit)
```