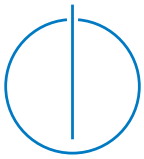
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Predicting Enterprise Application Performance Measures through Time Series Forecasting

Daniel Elsner



DEPARTMENT OF INFORMATICS

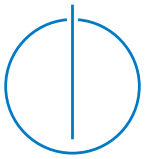
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Predicting Enterprise Application Performance Measures through Time Series Forecasting

Zeitreihenanalysen zur Vorhersage der Performanz von betrieblichen Anwendungen

Author:	Daniel Elsner
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Pouya Aleatrati Khosroshahi, M. Sc.
Submission Date:	April 15, 2018



I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, April 15, 2018 Daniel Elsner

Abstract

The reign of digital services and products requires continuous monitoring of the performance and availability of Information Technology (IT) enterprise applications. Today, inferior user experience such as slow server response times on websites (i.e., long waiting times) lead to worse conversion rates and thus diminish business success. By capturing essential operational measures around-the-clock, Application Performance Management (APM) enables practitioners to reactively detect and resolve performance regressions or abnormal system behavior. To increase the business value of enterprise applications, prior research emphasizes a shift from reactive towards proactive or predictive APM. Potential advantages are more efficient IT resource planning and broader analytical understanding of performance bottlenecks and anomalies.

The objectives of this thesis thus constitute the change of paradigm in APM to (i) forecasting performance regressions, and (ii) detecting abnormal system behavior in enterprise applications. The contribution is a set of developed models leveraging Machine Learning (ML) techniques for time series forecasting and anomaly detection.

To forecast performance regressions (i.e., high response times), we employ linear and nonlinear supervised ML algorithms on uni- and multivariate feature sets. Models with higher accuracy harness the nonlinear techniques, random forest regressors and Recurrent Neural Networks (RNNs). Moreover, the process of feature selection and engineering reveals that multivariate features sets (i.e., multiple APM measures) are superior to univariate modeling, which only considers historical response time.

In this thesis, we introduce a novel approach for the detection of abnormal system behavior based on density-based clustering. Our model detects abnormal system behavior more reliable than commonly used outlier detection techniques, which we adduce as baseline models. Additionally, the outlined approach provides an indication for the reason of the detected abnormal system behavior and hereby facilitates root cause analysis.

We evaluate the implemented models with real-world monitoring data from two productively running enterprise applications. The data set is obtained from a German car manufacturer and the scope of the analysis is defined in coordination with APM domain experts.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
APM	Application Performance Management
ARIMA-BP	Auto-Regressive Integrated Moving Average with Back-Propagation
ARIMA	Auto-Regressive Integrated Moving Average
ART	Average Response Time
AUC	Area Under The Curve
CART	Classification and Regression Trees
CPU	Central Processing Unit
CRISP-DM	Cross-Industry Standard Process for Data Mining
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DG	Design Guideline
DIRMO	Direct Multi-Output
DL	Deep Learning
DS	Design Science
EAM	Enterprise Architecture Management
EA	Enterprise Architecture
FN	False Negatives
FPR	False Positive Rate
FP	False Positives
GRU	Gated Recurrent Unit
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IS	Information Systems
IT	Information Technology
JMX	Java Management Extensions
JVM	Java Virtual Machine
kNN	k-Nearest Neighbor
LOF	Local Outlier Factor
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MIMO	Multi-Input Multi-Output

MLP	Multilayer Perceptron
ML	Machine Learning
MSE	Mean Squared Error
NAR	Nonlinear Auto Regressive
NMF	Non-Negative Matrix Factorization
PCA	Principal Component Analysis
PCC	Pearson Correlation Coefficient
QoS	Quality of Service
RBS	Recursive Binary Splitting
RC	Research Contribution
ReLU	Rectified Linear Units
REST	REpresentational State Transfer
RMSE	Root Mean Squared Error
RMSLE	Root Mean Squared Logarithmic Error
RMSPE	Root Mean Squared Percentage Error
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RQ	Research Question
RSS	Residual Sum of Squares
SLA	Service Level Agreement
SQ	Search Query
SQL	Structured Query Language
SSE	Sum of Squared Errors
SVM	Support Vector Machine
TN	True Negatives
TOGAF	The Open Group Architecture Framework
TPR	True Positive Rate
TP	True Positives
UI	User Interface
WUM	Web Usage Mining
XML	Extensible Markup Language

1. Introduction & Motivation

In the beginning of the 21st century, a vast amount of organizations are offering services and products through online platforms. It therefore becomes difficult to disentangle digital products from underlying IT infrastructures, which continually spurs the development of a digital economy (Bharadwaj et al., 2013). Consequently, the performance and availability of these IT services have high business impact. Today, poor user experience such as slow server response times on websites (i.e., long waiting times) lead to worse conversion rates and thus diminish business success. For example, by 2006 Google[1] already lost 20% user traffic if their websites responded 500 ms slower (Linden, 2006). To address these problems and to support expanding business processes, enterprise applications (i.e., IT systems) have become more complex regarding their architecture and implementation. In order to still adequately manage the growing number of deployed enterprise applications, APM systems are established to continuously monitor the Quality of Service (QoS). By capturing essential operational measures, e.g., response time or amount of requests, enterprise applications' reliability and the quality of user experience is quantified. Thereby, IT practitioners are able to keep track of time periods of performance regressions, e.g., high response times, or abnormal system behavior, e.g., service unavailable. Ergo, the goal of APM is to provide data-driven insights into the operational performance of enterprise applications and their infrastructure, and thus potentially improve user experience and thereby ensure alignment of interests between business and IT.

1.1. Towards Proactive Application Performance Management

APM is a discipline at the core of IT operations, which has the objective to continuously monitor relevant performance measures in enterprise applications (Heger et al., 2017). Activities in APM typically encompass collection, storage, presentation, and interpretation of monitoring data. By conducting the latter in an iterative manner, issues related to performance, availability, or business logic may be detected and resolved during the operation of an enterprise application (Heger et al., 2017). Despite that these derived actions are indeed valuable, they are reactive by nature. For example, a system alert that is raised because the response time exceeded a defined threshold is still activated too late to prevent the performance regression. Similarly, if an enterprise

[1] `https://www.google.com`, accessed on March 29, 2018

application behaves peculiarly, i.e., abnormally, this first has to be detected, before the actual process of manually finding the root cause and fixing the issue can begin. To approach this problem of reactivity and to increase the business value of enterprise applications, Panwar (2013) emphasizes a shift from reactive monitoring towards proactive or predictive APM. Potential advantages are more efficient IT resource planning and broader analytical understanding of performance bottlenecks and anomalies (i.e., patterns, which do not reflect observed normal behavior). These might result in less operational costs and higher customer satisfaction, which both ultimately lead to greater revenues. Hence, the proposed shift can be further narrowed down to two objectives: (i) forecasting performance regressions, and (ii) detecting abnormal system behavior in enterprise applications. The accomplishment of these objectives is yet contingent on the extent to which monitoring data (i.e., APM measures) – typically stored in time series or traces data (see section 2.1) – can be harnessed for predictive modeling.

Predicting future developments is a challenging task and relevant for almost all fields of science, engineering, and business administration (Palit and Popovic, 2005). Recently, methodologies from ML, statistical learning, or Artificial Intelligence (AI) are extensively discussed and applied in time series forecasting tasks (Bontempi et al., 2013). The detection of outliers or anomalies similarly opens a wide spectrum of modeling and ML techniques (Chandola et al., 2009). Throughout the course of this thesis, we[2] examine these methodologies, as we aim to compile models for the prediction of APM measures and detection of abnormal system behavior in enterprise applications.

1.2. Research Questions & Contributions

Based on the motivated shift towards proactive and analytical APM, there are two objectives of this thesis: (i) to evaluate different approaches to forecast performance, and (ii) to detect abnormal system behavior in enterprise applications. Hence, we strive to explore the following Research Questions (RQs):

- **RQ_1:** What are research gaps in existing scientific work regarding enterprise application performance forecasting and detection of abnormal system behavior in enterprise applications?
- **RQ_2:** Which are relevant APM measures to assess enterprise application performance ($RQ_{2.1}$) and how well can the identified be predicted based on historical APM measures ($RQ_{2.2}$)?
- **RQ_3:** How well can abnormal system behavior of enterprise applications be detected based on APM measures?

[2]The author Daniel Elsner and advisor

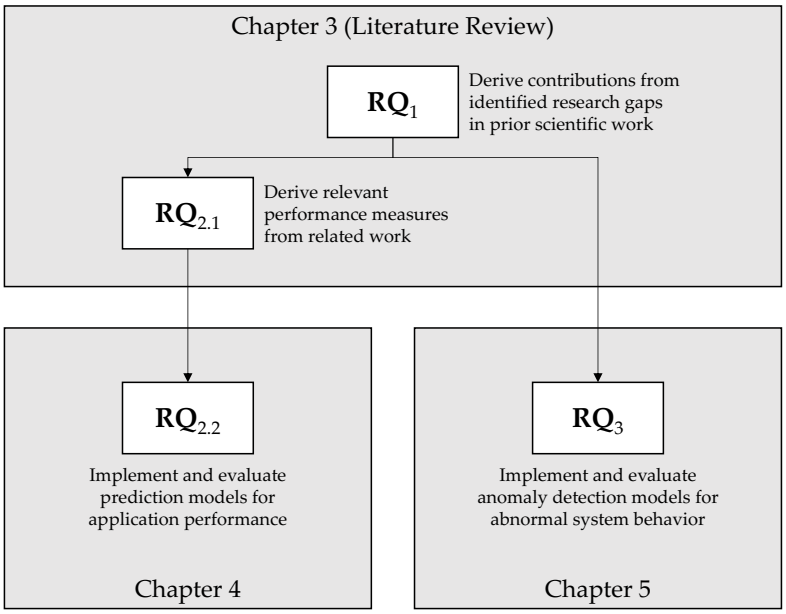


Figure 1.1.: Causal and sequential dependencies of the RQs and their assignment to chapters

Figure 1.1 describes the causal and sequential dependencies of the RQs. Moreover, it reasons splitting RQ_2 into two consecutive RQs as they are being covered in different chapters. RQ_1 and $RQ_{2.1}$ require a thorough literature review, which is conducted in chapter 3. Hereby, based on related scientific work and the results from RQ_1, the two Research Contributions (RCs) of this thesis covered in chapters 4 and 5 are further specified as:

- **RC_1:** Implementation and comparison of the linear and nonlinear ML techniques for multivariate multi-step time series application performance forecasting: linear regression, tree-based regressors, and neural networks.
- **RC_2:** Implementation of a multivariate density-based anomaly detection model providing an indication for the root cause of abnormal system behavior.

Both of them are evaluated on two productively running enterprise applications using real monitoring data collected by an industry partner, a German automobile manufacturer.

1.3. Research Approach

In Information Systems (IS) research, the Design Science (DS) paradigm is a prevalent research approach to "extend the boundaries of human and organizational capabilities

by creating new and innovative artifacts" (Hevner et al., 2004). By answering the described RQs and developing RCs, we seek to increase the business value of enterprise applications. Peffers et al. (2007) propose a research methodology, which serves as a template conducting DS research and successfully create and evaluate research artifacts. The conceptual framework of our research is therefore established on guidelines from Hevner et al. (2004) and Peffers et al. (2007). Figure 1.2 summarizes the research approach from the adapted DS paradigm. It emphasizes an initial rigorous theoretical foundation before developing the contributions. The problem domain is further sliced into workable packages, and the two research artifacts are developed with iterative feedback loops.

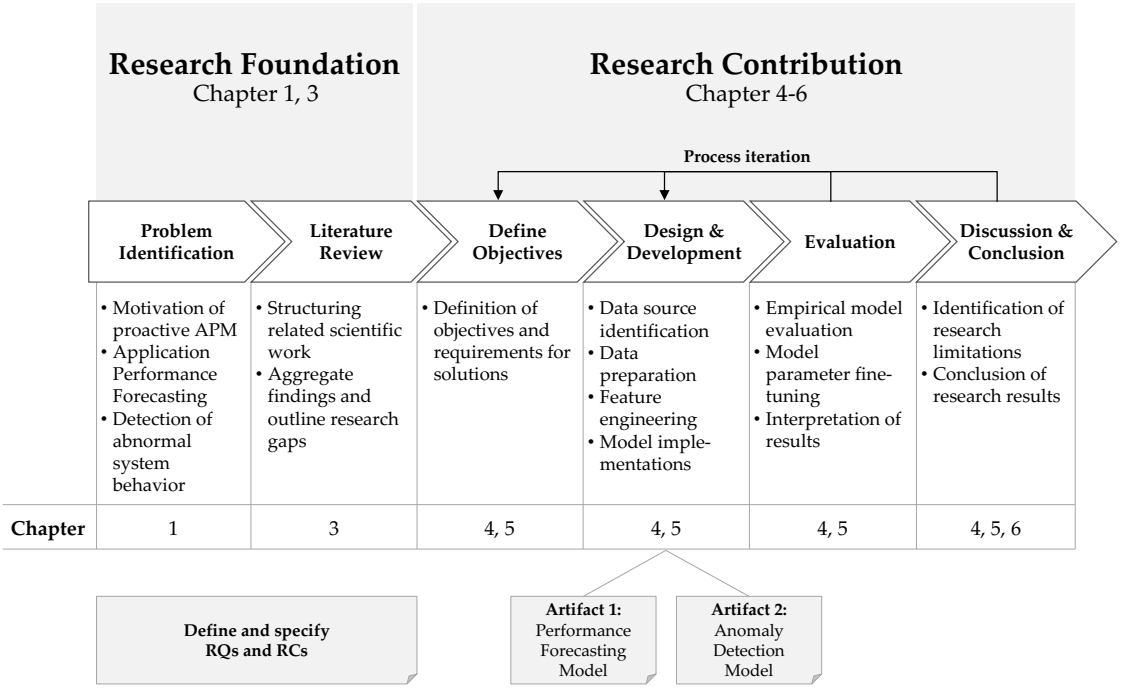


Figure 1.2.: Research approach motivated by Hevner et al. (2004) and Peffers et al. (2007) capturing DS paradigms

The research approach contains the following six main steps:

1. **Problem Identification:** The problem domain is outlined and the research focus on application performance forecasting and detection of abnormal system behavior is motivated.
2. **Literature Review:** Theoretical foundations are established and related scientific work is reviewed. Subsequently, the contributions RC_1 and RC_2 are constituted.
3. **Define Objectives:** Objectives and requirements for developed research contributions are defined. This is done in the beginning of the respective contribution chapters.

4. **Design & Development:** This step encompasses the process of gathering and preparing monitoring data as well as the implementation of two research artifacts. These are characterized by selecting from suitable learning algorithms and time series analysis techniques.
5. **Evaluation:** The evaluation step comprises a quantitative evaluation of developed models and iterative model tuning afterwards.
6. **Discussion & Conclusion:** In the last step, results from the developed models are interpreted and limitations are discussed. These insights are incorporated into step 3 and 4 iteratively to ensure relevancy of results.

The outline of the thesis is defined by the research approach: First, we introduce the theoretical background behind APM, Enterprise Architecture Management (EAM), time series analysis, and anomaly detection in chapter 2. Second, a literature review is conducted to identify related work and derive the contributions of this thesis in chapter 3. Third, chapter 4 presents RC_1, a comparison of linear and nonlinear ML techniques applied to the time series forecasting problem of APM data. Fourth, chapter 5 analogously contains RC_2, an approach towards density-based detection of abnormal system behavior in enterprise applications. Both of the aforementioned chapters include a rigorous evaluation and interpretation of the results. Last, chapter 6 concludes the contributions of this thesis, and suggests future research based on identified limitations.

2. Foundations

The following sections will establish fundamental technologies relevant throughout the course of this thesis. The introduced topics are APM, EAM, methodologies of time series analysis and ML, and anomaly detection. The purpose of this chapter is to consolidate an understanding of related research areas and techniques applied to answer the previously formulated RQs.

2.1. Application Performance Management

As outlined in section 1.1, the performance of enterprise applications has high business impact. An example for poor performance is high server response time on websites, i.e., long waiting times for potential customers. By 2008, Amazon[1] lost 1% of sales revenue for every 100 ms of latency (Liddle, 2008). APM has the objective to monitor relevant performance and infrastructure measures during the operation of IT applications. It comprises tools and techniques for continuously tracking the application state, and for detecting and resolving performance-related issues by analyzing the collected monitoring data (Heger et al., 2017).

2.1.1. APM Activities

Generally, APM systems implement four main iterative activities. Namely, data collection, data storage and processing, data presentation and data interpretation and use (Heger et al., 2017) (see figure 2.1).

- **Data Collection:** As modern applications' architectures are often highly distributed (i.e., multiple physical tiers), multi-layered and used by heterogeneous clients (e.g., web client, mobile app client), monitoring data needs to be collected on multiple layers and system tiers. Therefore, an end-to-end APM requires the collection of measures on multiple physical locations over multiple layers (e.g., presentation, business and persistence layer) and system levels (e.g., middleware, operating system, hardware). Usually, so-called monitoring agents, a software component, collect the measures on the monitored system either passively when certain events occur, or actively or periodically.

[1] `https://www.amazon.com/`, accessed on March 28, 2018

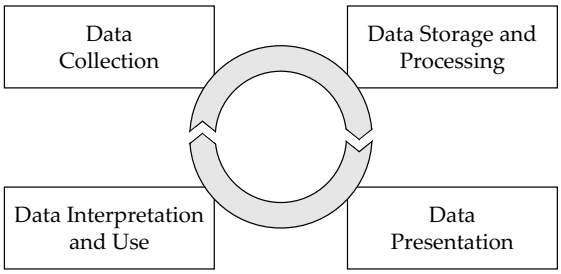


Figure 2.1.: Four main iterative activities implemented by APM systems (Heger et al., 2017)

- **Data Storage and Processing:** To make collected monitoring data usable for analyses, it needs to be stored into higher-level data structures. Different approaches are storing data into time series, execution traces or both. As detailed trace data can easily develop into a large data set, it is often only kept in the short term (whereas summary statistics are kept in the long term in time series).
- **Data Presentation:** Data can be understood through meaningful and comprehensive visualizations on different levels of abstraction and detail. Thus, APM data needs to be made accessible through intuitive visualizations by purposefully selecting dimensions and aggregation metrics that can provide expressive information about an application's health.
- **Data Interpretation and Use:** The interpretation of available monitoring data aims to detect problems and couple them with a notification/alert functionality, to conduct an automatic and manual root cause analysis of occurred problems, and to implement an automatic system adaption to performance issues, e.g., auto-scaling in cloud environments (Lorido-Botran et al., 2014).

2.1.2. Monitoring Software

From feature-rich APM tool suites to dedicated tools for collecting specific singular measures, there exists a large amount of software to implement the aforementioned APM activities. According to Gartner, the most mature software solutions are commercial products such as AppDynamics[2], CA APM[3], Dynatrace[4], and New Relic[5] (Haight et al., 2015). Commonly used open-source tools are Nagios[6] for monitoring on system-level,

[2] `https://www.appdynamics.com/`, accessed on March 28, 2018
[3] `https://www.ca.com/`, accessed on March 28, 2018
[4] `https://www.dynatrace.com/`, accessed on March 28, 2018
[5] `https://newrelic.com/`, accessed on March 28, 2018
[6] `https://www.nagios.org/`, accessed on March 28, 2018

inspectIT[7] or Pinpoint[8] (only Java, i.e., Java Virtual Machine (JVM)) for application-level monitoring, and Zipkin[9] for distributed execution traces. But also the Java Management Extensions (JMX)[10] allow to monitor Java applications through a respective interface. Ahmed et al. (2016) provide an overview and a comparison regarding ease of installation and approaches for detecting performance anomalies of most of these APM tools. Additionally, aggregated time series are often stored into time series databases such as Graphite[11] or InfluxDB[12], which are then visualized on dashboards such as Grafana[13].

Typically, APM software supports resource and (web) traffic monitoring, automatic architecture and component discovery, and end-user experience monitoring. From this data, aggregation measures (e.g., count, mean, median) are calculated and visualized, and anomalies are detected through self- or system-defined thresholds. Usually, more complex trace data can be analyzed and drilled down through several predefined views, e.g., method call stack or transaction flow. Mature tools are in general programming language agnostic or at least support a wide range of languages.

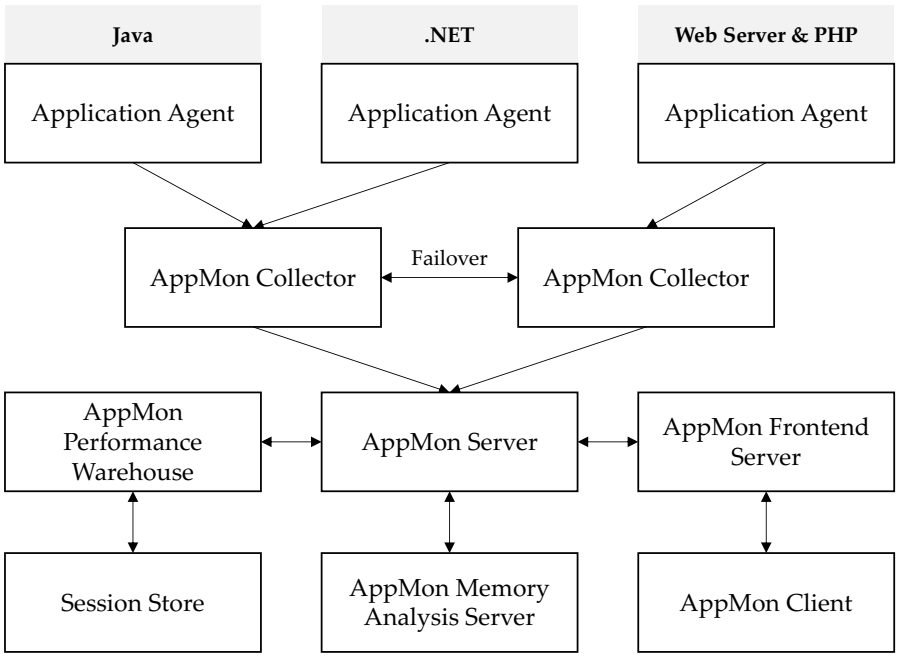


Figure 2.2.: Architecture of Dynatrace AppMon environment

[7] http://www.inspectit.rocks/, accessed on March 28, 2018
[8] https://github.com/naver/pinpoint, accessed on March 28, 2018
[9] https://zipkin.io/, accessed on March 28, 2018
[10] https://docs.oracle.com/javase/tutorial/jmx/, accessed on March 28, 2018
[11] https://graphiteapp.org/, accessed on March 28, 2018
[12] https://www.influxdata.com/, accessed on March 28, 2018
[13] https://grafana.com/, accessed on March 28, 2018

Exemplary, the Dynatrace monitoring solution AppMon is further investigated, figure 2.2 illustrates its architecture[14]. Core components are an AppMon server, AppMon memory analysis server, AppMon frontend server, AppMon collector, AppMon client, application agents, and a AppMon performance warehouse database:

- **Application Agents:** Agents are lightweight components for specific technologies, e.g., Java, .NET, which are injected into the application layer as processes or daemons that retrieve data from the application instance and send them to nearby AppMon collectors.
- **AppMon Collector:** The collector correlates and bundles the data and sends them to the AppMon server.
- **AppMon Server:** The AppMon server then correlates the so-called PurePath data – execution traces information – and stores it into the performance warehouse database, together with thread and memory dumps that are stored into a session store.
- **AppMon Frontend Server:** In order to provide reliable and quick data analysis to clients, the frontend server handles respective requests from AppMon clients.
- **AppMon Memory Analysis Server:** The memory analysis server is responsible for post-processing memory snapshots to offload resource usage from AppMon servers.
- **AppMon Performance Warehouse:** The performance warehouse (Structured Query Language (SQL) database server) stores processed measure/time series data from agents. As opposed to time series data, PurePaths and thread and memory dump are directly stored in the session store.
- **AppMon Client:** AppMon client is a software installed on users' machines or a web user interface, which accesses the frontend server to receive data to visualize analyses on monitoring data.

Even though problem detection, diagnosis and prediction are among the goals of the data interpretation activity of monitoring tools, they are often still limited to their alerting and visualization functions. Additionally, APM software most commonly is not meant to be extended. This makes it difficult to make further root cause analyses or implement more advanced anomaly detection approaches than the usually rather basic ones (e.g., for Dynatrace, 90th percentile as anomaly threshold) built into the tool already (Ahmed et al., 2016). As a result, root cause analysis and prediction of application performance requires expert knowledge and manual efforts (Heger et al., 2016). Heger et al. (2017) find that "systematization of expert knowledge and ML approaches could provide key support here" (Heger et al., 2017). In section 2.3 and 2.4, we elaborate on techniques

[14] Based on AppMon Platform Overview – Architecture, `https://community.dynatrace.com/community/display/DOCDT65/Architecture`, accessed February 28, 2018

that can possibly harness the potential of monitoring data collected by APM tools to detect anomalies and predict application performance in advance.

2.1.3. Performance Measures

From the large variety of collected data, there are multiple measures (i.e., metrics, monitors) that could help detect application performance regressions. $RQ_{2.1}$ requires the investigation of previous work done in APM, and the prediction of application performance to identify the relevant APM measures assessing application performance. In APM research, Rabl et al. (2012), Wert et al. (2013), and Ahmed et al. (2016) see (server) response time as an effective measure to evaluate application performance. Besides response time, Brunnert and Krcmar (2015) consider throughput and resource utilization as indicators for system (i.e., application) performance. In the area of cloud computing, Service Level Agreement (SLA) are also often based on response time and throughput for monitoring cloud servers (Patel et al., 2009). As APM is heavily influenced by non-scientific work done by domain practitioners, it is important to take these sources into account. Articles by Appoptics (Goldberg, 2017), Stackify (Watson, 2017), APMDigest (Tunis, 2016), and CDNnetworks (Bell, 2015) outline the top five APM measures in table 2.1 (sorted by relevance).

Table 2.1.: Top five APM measures to assess enterprise application performance from APM domain practitioners

APM Measure	Count	Sources
Response Time (Average, Peak/Maximum)	4	Goldberg, 2017; Bell, 2015 Watson, 2017; Tunis, 2016
Request Load (Requests per Minute/Amount Requests)	4	Goldberg, 2017; Bell, 2015 Watson, 2017; Tunis, 2016
Error Rate (Hypertext Transfer Protocol (HTTP) Errors, Exception Count)	3	Goldberg, 2017; Bell, 2015 Watson, 2017
Resource Utilization (Central Processing Unit (CPU), Database)	2	Goldberg, 2017 Watson, 2017
Concurrent Users	1	Bell, 2015

2.2. Enterprise Architecture Management

Enterprise Architecture (EA) emphasizes a rigorous connection between business and IT in the holistic modelling, integrated planning, and implementation of corporate IT

environments (Aier et al., 2011). Accordingly, architecture can be understood as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" (IEEE-SA Standards Board, 2000). Moreover, The Open Group Architecture Framework (TOGAF) describes architecture as "formal description" to guide implementation of a system and structure interrelationships and dependencies of its components, as well as "governing their design and evolution over time" (The Open Group, 2009).

In a rapidly changing environment of customer requirements and business objectives, EAM principles can support organizational change and increase agility by incorporating both, business-related and IT artifacts (Stelzer, 2010; Kaisler et al., 2005). EAM, as a strategic management function, is concerned with describing, analyzing, and communicating the "current, planned, as well as the envisioned target state of the EA" (Buckl, 2011). To reduce complexity and enable manageability of EA, the description or modelling typically conceptually divides EA into layers or architecture domains, i.e., subsets of EA. TOGAF distinguishes between (i) business architecture, (ii) data architecture, (iii) application architecture, and (iv) technology architecture (The Open Group, 2009). Another common approach from Buckl et al. (2011) defines three architectural layers, namely (1) business capabilities, (2) business services, and (3) infrastructure services (see figure 2.3).

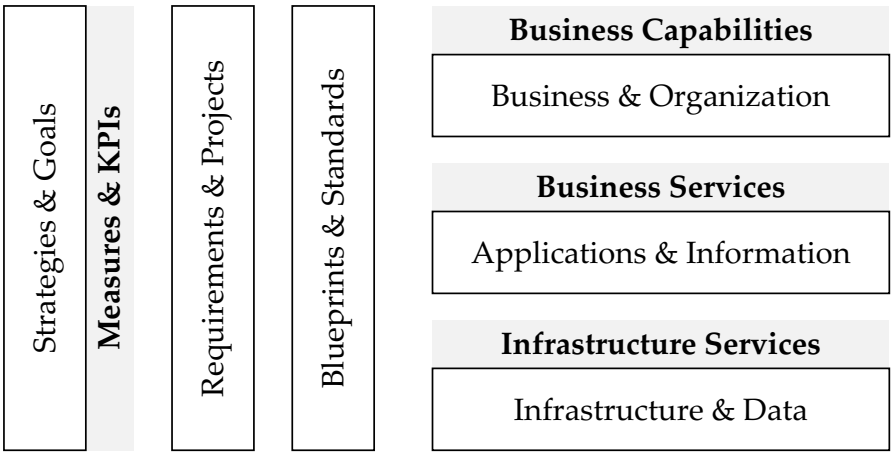


Figure 2.3.: Architectural layers of an EA according to Buckl et al. (2011)

As this thesis addresses enterprise applications' and their infrastructures' performance, we identify (iii) application architecture and (iv) technology architecture, respectively (2) business services and (3) infrastructure services, as the relevant domains. Furthermore, Braun and Winter (2005) understand the application landscape, i.e., a set of enterprise applications and their inter-dependencies, as an integral part of the EA. By incorporating both of these understandings of EA domain separation, we narrow the focus of this thesis down to the investigation of performance drivers and IT infrastructure of enterprise applications. Thereby, the insights derived from our analyses can help improving the

data/information flow between the EA application and infrastructure layers in the following ways:

- **Identification of Architectural Performance Anti-Patterns:** By analyzing performance bottlenecks or system outages, patterns could be identified on EA level where undesired (abnormal) system behavior in enterprise applications correlate with the IT infrastructure.
- **Optimization of Architectural Planning:** By getting to know the factors, which drive operational performance, resource and architectural planning can be optimized and applications can be compared regarding their performance profiles.
- **Increase Business Value of Application Landscape:** As aforementioned, a shift from a reactive towards a proactive application/infrastructure maintenance (i.e., APM) promises higher business value of the application landscape (Panwar, 2013).
- **Leverage Synergies in EA:** Analogical trends in APM measures can reveal similarities in data access, functional reuse or ownership support of enterprise applications. By analyzing these, the EA can be cleaned, improved (e.g., standardized), and aligned more thoroughly with business requirements (Buckl, 2011; Braun and Winter, 2005; Van Der Raadt and Van Vliet, 2008).
- **Increase IT Agility:** MacCormack et al. (2017) find that IT agility – "the ability to rapidly update, improve, remove, replace, and re-imagine the software applications that underpin a firm's competitive position"(MacCormack et al., 2017) – correlates with different types of couplings between software applications (e.g., directly coupled, indirectly coupled). Thus, couplings or dependency loops detected through the analysis of APM measures (e.g., nodes visited per execution trace) may indicate decreasing IT agility.

2.3. Time Series Analysis

Time series are series of data points, which are collected and indexed in time order. Typical examples for such sequence data are customer data, e.g., purchase history of a given customer, web server logs, e.g., browsing activity of a particular web visitor, or sensor data, e.g., history of alarm events generated by a given sensor (Bontempi et al., 2013). Generally, time series analysis comprises methodologies for analyzing these time series, whereas time series forecasting employs an abstract model (e.g., statistical or mathematical model) to derive predictions for future values based on observations from previous time steps.

One of the objectives of this thesis is the prediction of future performance of enterprise applications from APM monitoring data (e.g., server response times) stored in time series. In section 3.2, we outline the research gap and define the applied methodologies

within this thesis: linear regression, tree-based regressors, and neural networks. Hence, the following methods among numerous existing approaches for time series forecasting are reviewed:

- **Linear Statistical Methods**: Since decades the time series forecasting domain has been influenced by linear statistical methods from econometrics and other fields of science and engineering (e.g., finance, meteorology, telecommunication) (Bontempi et al., 2013; Palit and Popovic, 2005). The most prominent representatives are Auto-Regressive Integrated Moving Average (ARIMA) models. However, their assumption of linear relationships between lagged (i.e., prior) and current values within the time series is a restriction which makes them impracticable for a large field of nonlinear applications (De Gooijer and Hyndman, 2006).
- **Machine Learning**: ML models have increasingly established themselves in time series forecasting in the past years (Bontempi et al., 2013; Msiza et al., 2007). Parametric and non-parametric nonlinear ML models such as simple Artificial Neural Networks (ANNs), k-Nearest Neighbors (kNNs), or tree-based regressors have outperformed classical statistical (linear) methods in various tasks (Bontempi et al., 2013; Ahmed et al., 2010).
- **Deep Learning**: Recently, more advanced ANNs with specific architectures, i.e., deep and recurrent neural networks, are becoming more popular as they have proven to be suitable for sequence-to-sequence time series prediction (Din and Marnerides, 2017; Längkvist et al., 2014). This domain, which is often referred to as Deep Learning (DL), allows the modelling of memory into neural networks, hence information can be considered by the model over arbitrary time intervals.

Regardless of the exhaustive research done in this area, linear statistical models do not exactly match the requirements of analyses performed within this thesis. As in APM typically multiple variables are monitored, we require extensibility of models to multi-dimensional inputs, i.e., consideration of multivariate time series. Classical linear methods need to be specifically adapted to address this need (Aboagye-Sarfo et al., 2015). Additionally, they are – as previously outlined – unsuitable for modeling nonlinear dependencies. Nor do they allow online learning (i.e., updating model parameters once more data becomes available) out of the box, which makes them inefficient for handling large-scale data sets in real-time (Liu et al., 2016). Since ML and DL algorithms are proficient in modelling nonlinear relationships and using multivariate time series as input, the analyses in chapter 4 compare the predictive performance of the latter and modest multivariate linear regression.

Naturally, planning tasks, which rely on the forecast of a time series, can heavily benefit if a model accurately predicts more than one future time step. However, the choice of the size of the forecasting period is a non-trivial task: one-step forecasting is challenging, but multi-step forecasting models are even more difficult to model and to fit to training data. Existing approaches for multi-step forecasting either accumulate prediction errors,

reduce predictive accuracy, or increase uncertainty of the prediction (Ben Taieb et al., 2010; Ben Taieb et al., 2012).

The next subsection 2.3.1 discusses these approaches, formally introduces to time series modeling, and describes the process of reframing (i.e., pre-processing) time series into supervised learning problems.

2.3.1. Forecasting Problem Modeling

A time series can be formally described as a collection of N observations over time, $Y_N = y_1, ..., y_N$. The goal of time series forecasting is to forecast a sequence of h future values $y_{t+1}, ..., y_{t+h}$ of this time series at an arbitrarily chosen point in time t. Dynamical systems theory interprets a time series as the observable of a dynamic system with state $s(t)$, denoting the value of the state at time t (Casdagli et al., 1991; Bontempi et al., 2013). Accordingly, in order to reconstruct an adequately equivalent state, i.e., reconstruct (i.e., predict) a time series value y_{t+1}, the only available information is contained in the t prior observations $y_1, ..., y_t$.

Hence, for a one-step ahead prediction of a future time step y_{t+1} we can derive:

$$y_{t+1} = f(y_t, y_{t-1}, ..., y_{t-d+1}) + w \tag{2.1}$$

with $t \in \{d, ..., N-1\}$.

This representation implies that any forecasted time series value relies on a set of d (often called lag, order or embedding dimension parameter) prior observations from the time series. Often the vector $y_t, y_{t-1}, ..., y_{t-d+1}$ is called the feature vector – denoted by X – and each element of the vector can be interpreted as a predictor of y_{t+1}. The estimator f denotes the functional temporal dependency between observations whose parameters are learned in the so-called model training phase. However, as we assume that we will not be able to learn a perfectly accurate model of f, we include the term w, which contains "modeling error, disturbances and/or noise" (Ben Taieb et al., 2012). Bontempi et al. (2013) refer to this equation as "a statistical Nonlinear Auto Regressive (NAR) formulation" (Bontempi et al., 2013) of the time series forecasting problem.

As aforementioned, the ultimate goal is to generate multiple-step ahead forecasts, i.e., learn from data multi-input multi-output dependencies to make sequence-to-sequence predictions. Ben Taieb et al. (2012) give an overview of the five prevalent H-step ahead strategies proposed in time series research, namely Recursive, Direct, DirRec, Multi-Input Multi-Output (MIMO), and Direct Multi-Output (DIRMO) strategy. H can be interpreted as the time horizon that is to be predicted. Depending on the chosen strategy one or multiple models depicted by f are trained with different shortcomings and benefits (see table 2.2):

- **Recursive Strategy**: The most intuitive strategy is to have H recursive one-step ahead forecasts where the result of the previous one-step forecast is fed into the next one-step forecast as an additional predictor in the feature vector.

$$\hat{y}_{N+h} = \begin{cases} \hat{f}(y_N, ..., y_{N-d+1}) & \text{if } h = 1 \\ \hat{f}(\hat{y}_{N+h-1}, ..., \hat{y}_{N+1}, y_N, ..., y_{N-d+h}) & \text{if } h \in \{2, ..., d\} \\ \hat{f}(\hat{y}_{N+h-1}, ..., \hat{y}_{N+h-d}) & \text{if } h \in \{d+1, ..., H\} \end{cases} \tag{2.2}$$

  Consequently, the prediction error accumulates over time which is the major limitation of this strategy (Cheng et al., 2006; Hamzaçebi et al., 2009).

- **Direct Strategy**: The direct strategy implies the training of H independent models f_H, one for each time step in the forecasting horizon H:

$$\hat{y}_{N+h} = \hat{f}_h(y_N, ..., y_{N-d+1}). \tag{2.3}$$

  Even though this prevents prediction error accumulation, complex dependencies between predictions $\hat{y}_{N+h}$ cannot be considered, nor is it computationally inexpensive as H models need to be trained (Cheng et al., 2006; Hamzaçebi et al., 2009).

- **DirRec Strategy**: This strategy combines the latter two strategies by learning H models f_H, but additionally incorporating forecasts from the previous steps. Note that the feature vector size (i.e., embedding size d) increases over the forecasting horizon.

$$\hat{y}_{N+h} = \begin{cases} \hat{f}_h(y_N, ..., y_{N-d+1}) & \text{if } h = 1 \\ \hat{f}_h(\hat{y}_{N+h-1}, ..., \hat{y}_{N+1}, y_N, ..., y_{N-d+1}) & \text{if } h \in \{2, ..., H\} \end{cases} \tag{2.4}$$

  Despite its alleged performance, this strategy has not yet been investigated much in research (Sorjamaa and Lendasse, 2006).

- **MIMO Strategy**: The MIMO strategy, in contrast to the previously mentioned strategies, which imply single-output functions, learns one multiple-output model F

$$[\hat{y}_{t+H}, ..., \hat{y}_{t+1}] = \hat{F}(y_t, ..., y_{t-d+1}) \tag{2.5}$$

  with $t \in \{d, ..., N-H\}$, a vector-valued function $F : \mathbb{R}^d \to \mathbb{R}^H$ (Micchelli and Pontil, 2005; Ben Taieb et al., 2012). The strategy has been successfully applied in various forecasting tasks, however, using one model to forecast all the horizons $h \in \{1, ..., H\}$ could reduce flexibility and adjustability of the forecasting approach compared to multiple models (Bontempi, 2008; Ben Taieb et al., 2010).

- **DIRMO Strategy**: To combine direct and MIMO strategy, DIRMO creates n forecasting blocks from the horizon H defined by an integer parameter s, i.e., $n = \frac{H}{s}$ blocks, where each block is forecasted using the MIMO strategy (Taieb et al., 2009). The strategy returns H forecasts by estimating n models F_p:

$$[\hat{y}_{N+p*s}, ..., \hat{y}_{N+(p-1)*s+1}] = \hat{F}_p(y_N, ..., y_{N-d+1}) \tag{2.6}$$

with $t \in \{d, ..., N - H\}$, $p \in \{1, ..., n\}$, and a vector-valued function $F_p : \mathbb{R}^d \to \mathbb{R}^s$ if $s > 1$. By gradually adjusting s, the most suitable forecasting horizon block size and thus, the best trade-off between direct and multi-output prediction can be found. Ben Taieb et al. (2010) have applied the DIRMO strategy successfully to forecasting competition data sets.

Table 2.2.: Pros and cons of the different multi-step forecasting strategies (Ben Taieb et al., 2012)

	Pros	Cons	Computational Time Needed
Recursive	Suitable for noise-free time series (e.g., chaotic)	Accumulation of errors	+
Direct	No accumulation of errors	Conditional independence assumption	++++
DirRec	Trade-off between Direct and Recursive	Input set grows linearly with H	+++++
MIMO	No conditional independence assumption	Reduced flexibility: same model structure for all the horizons	++
DIRMO	Trade-off between total dependence and total independence of forecast	One additional parameter to estimate	+++

In chapter 4, we apply and compare the direct and the MIMO strategy for multi-step time series forecasting with different results.

The previous considerations were focused on the case of univariate time series. Generally, univariate time series analysis is a good starting point for creating baseline forecasting models. However, most of the times (and also in APM) multiple time series are collected continuously. By moving towards multivariate time series analysis, we can exploit new potentially reasonable predictors for forecasting future time steps of our time series Y_N (Goel et al., 2017; Aboagye-Sarfo et al., 2015). An example from weather forecasting is to predict the precipitation for the next day not only based on the previous week of daily precipitation observations (i.e., time series P), but also take into account the temperature observations (i.e., time series T) from the past week. More formally, if we consider three time series $A_N = a_1, ..., a_N$, $B_N = b_1, ..., b_N$, and $Y_N = y_1, ..., y_N$, we define the multivariate one-step ahead forecasting problem of a future time step y_{t+1} as:

$$y_{t+1} = f(y_t, y_{t-1}, ..., y_{t-d+1}, a_t, a_{t-1}, ..., a_{t-d+1}, b_t, b_{t-1}, ..., b_{t-d+1}) \tag{2.7}$$

The equation 2.7 implies that f will learn how significant dependency relationships are between Y_N, A_N, and B_N by considering prior time series values from all three series for predicting y_{t+1}. Similar to the univariate case it is possible to use one of the multi-step forecasting strategies to formulate a multi-step forecasting problem with horizon H.

Supervised learning is a sub-domain of ML where an algorithm is used to learn the mapping function from a set of input variables X to an output variable Y. Subsection 2.3.2 further investigates the separation into unsupervised and supervised learning, and regression and classification problems. In order to be able to learn models, which predict future time steps, we need to frame our previously derived uni-/multivariate forecasting problems into a supervised learning setting, more precisely into generic regression problems. Bontempi et al. (2013) propose a straightforward formulation for one-step forecasting which we extend to vectorial input (i.e., feature matrix) X and vectorial output (i.e., H forecasted time steps). X results into a $[(N-n-h)\times n]$ matrix, whereas Y is a $[(N-n-h)\times h]$ matrix.

$$X = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_2 & y_3 & \cdots & y_{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-n-h} & y_{N-n-h+1} & \cdots & y_{N-h} \end{bmatrix} \tag{2.8}$$

$$Y = \begin{bmatrix} y_{n+1} & y_{n+2} & \cdots & y_{n+h} \\ y_{n+2} & y_{n+3} & \cdots & y_{n+h+1} \\ \vdots & \vdots & \vdots & \vdots \\ y_{N-h+1} & y_{N-h+2} & \cdots & y_N \end{bmatrix} \tag{2.9}$$

Henceforth, we can use this supervised learning setting in the following chapters to learn regression models for forecasting (multiple) future time steps in uni-/multivariate time series analysis.

2.3.2. Machine/Deep Learning

Based on how they learn about data in order to make predictions, algorithms in ML can be classified into unsupervised and supervised learning (Jiawei et al., 2012; Hastie et al., 2001; James et al., 2013).

Supervised learning aims to learn the mapping function from a set of input variables X to an output variable Y. The learning process can be thought of as a teacher supervising his student: the algorithm iteratively makes predictions on training data and is corrected by the teacher until the algorithm achieves an acceptable level of prediction performance. Consequently, the algorithm's possible outputs are already known in advance and the data used to train the algorithm need to be labeled with correct answers. Supervised

learning problems are further split into classification and regression problems. Classification problems have a categorical output variable, e.g., disease and no disease or dog and cat in image recognition. Regression problems have an output variable which is a real value, e.g., dollars or temperature (Jiawei et al., 2012; Hastie et al., 2001; James et al., 2013).

Unsupervised learning has the goal to model the underlying structure or distribution of input data X, which does not have corresponding output variables. Thus, learning algorithms intend to discover data patterns or interesting – but often latent – structures. They can be be further divided into clustering and association algorithms. Clustering algorithms attempt to find inherent groupings in the data, e.g., grouping customers by their consumer behavior. Association rule mining algorithms discover rules that describe interesting relations between variables in the data, e.g., snow always comes with low temperature (Jiawei et al., 2012; Hastie et al., 2001; James et al., 2013).

Throughout this thesis, we examine different approaches of applying both, unsupervised and supervised learning techniques to the given time series forecasting problems. In the following paragraphs, the used supervised learning algorithms are briefly introduced, however for a more thorough explanation respective primary literature should be considered. Section 2.4 shines light on unsupervised learning techniques that have been applied for anomaly detection.

Linear Regression

The linear regression aims to model linear relationships between one or multiple input variables (i.e., features, explanatory or predictor variables) and a continuous target variable (i.e., response variable), e.g., sales of a company for a specific time period. The univariate linear regression assumes an approximate linear relationship of a single feature x to a target variable y,

$$y = w_0 + w_1 x \tag{2.10}$$

where w_0 is the Y-axis intercept and w_1 is the weight coefficient of x (James et al., 2013). The equation can be interpreted as the best-fitting straight line through n given observation pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The multivariate linear regression generalizes the regression model to multiple (m) feature variables:

$$y = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x \tag{2.11}$$

where w_0 is the y-axis intercept with $x_0 = 1$. Throughout the model training phase, estimates for w are produced (estimate for w is often denoted by $\hat{w}$) which incrementally

improve the model regarding the predictive quality for y (estimate for y is often denoted by $\hat{y}$).

Decision Tree & Random Forest

Classification and Regression Trees (CART) and their modern variations such as random forests are currently among the most powerful ML techniques (Breiman et al., 1984; Breiman, 2001). Generally, tree-based methods for regression and classification involve stratifying or segmenting the feature space into regions. The resulting set of splitting rules is summarized in a decision tree. Ensemble methods, e.g., random forests, produce multiple trees to yield a single consensus prediction with often improved prediction accuracy (James et al., 2013).

We focus on regression trees to create time series forecasting models for the previously introduced generic regression problems (see subsection 2.3.1). The process of building a regression tree involves dividing the feature space $X = X_1, X_2, \ldots, X_p$ into J distinct, non-overlapping regions $R_1, R_2, \ldots, R_J$ or more precisely tree leaves. Then, any newly available observation i with feature vector x_i falling into a region R_j receives as prediction $\hat{y}$ the mean of the target variable y of the training observations in R_j. The regions $R_1, \ldots, R_J$ are derived through an algorithm called Recursive Binary Splitting (RBS). The goal is to minimize the Residual Sum of Squares (RSS) for the regions given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{2.12}$$

where $\hat{y}_{R_j}$ denotes the mean value of y for the training observations within R_j. Due to the infeasibility of computation (np-complete decision problem) of an optimal partition set, RBS takes a top-down, greedy approach (James et al., 2013): At the tree root the feature space $X = X_1, \ldots, X_p$ is split into the regions

$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\}, \tag{2.13}$$

where j and s are chosen with respect to minimizing

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \tag{2.14}$$

leading to the greatest possible reduction in RSS. With the identified regions the process of binary splitting is repeated iteratively until a stopping criterion is reached, e.g., minimum amount of observations per leaf.

The resulting tree is easy to interpret, figure 2.4 shows an example of a binary regression tree for predicting the log salary of a baseball player, based on the number of years that

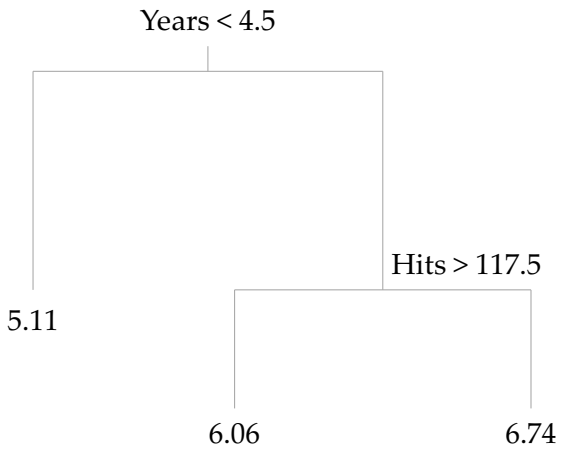


Figure 2.4.: Binary regression tree for predicting the log salary of a baseball player (James et al., 2013)

he has played in the major leagues and the number of hits that he made in the previous year (James et al., 2013). It is worth mentioning, that pruning – a technique where the maximal tree depth is limited – can reduce overfitting of trees to training data sets.

Random forests aggregate B decision trees which can substantially improve the predictive performance of the resulting model. The decision trees are trained on B bootstrapped training samples, meaning the training data is resampled from the original data sample. However, while building the tree, for each split we only consider a random sample of m predictors as split candidates from p available predictors. Typically, m is chosen as $m \approx \sqrt{p}$ which leaves us with a minority of available predictors to pick from at each split. In the end, we average the resulting prediction from the B constructed trees. Given that the decision trees are not pruned, they tend to grow deep with high variance and low bias. By averaging them the variance is reduced and hence predictive performance improved (Breiman, 2001).

Artificial Neural Networks

ANNs were originally designed based on concepts of the biological human brain and its interconnected nerve cells, neurons (McCulloch and Pitts, 1943; Rosenblatt, 1958). An artificial neuron can be interpreted as a simple function of its weighted inputs which generates an output. The function (f or ϕ) can either be a linear combination (i.e., perceptron (Rosenblatt, 1958)) of inputs or a different (nonlinear) activation function.

Figure 2.5 encapsulates the concept of the artificial neuron with where the sum of the weighted inputs is often called net input z. The most commonly used (nonlinear) activation functions are the logistic sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$), hyperbolic tangent

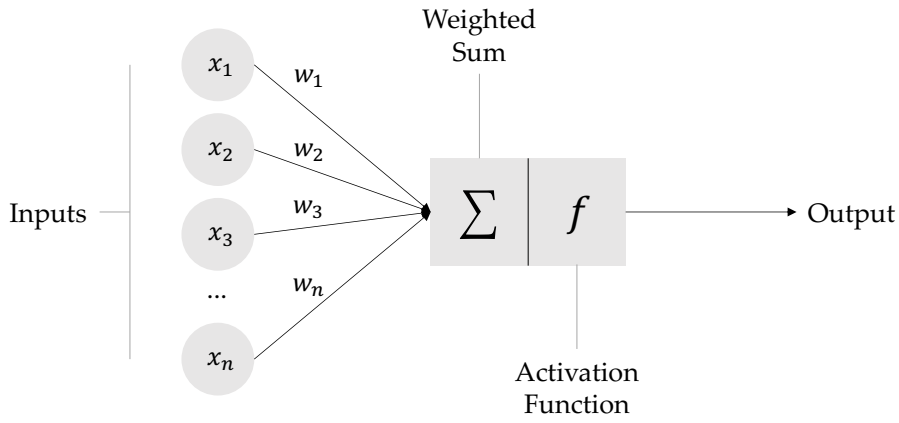


Figure 2.5.: Concept of an artificial neuron which generates an output based on weighted inputs

($tanh(x) = \frac{2}{1+e^{-2x}} - 1$), and Rectified Linear Units (ReLU) ($R(x) = \max(0, x)$) function (see figure 2.6).

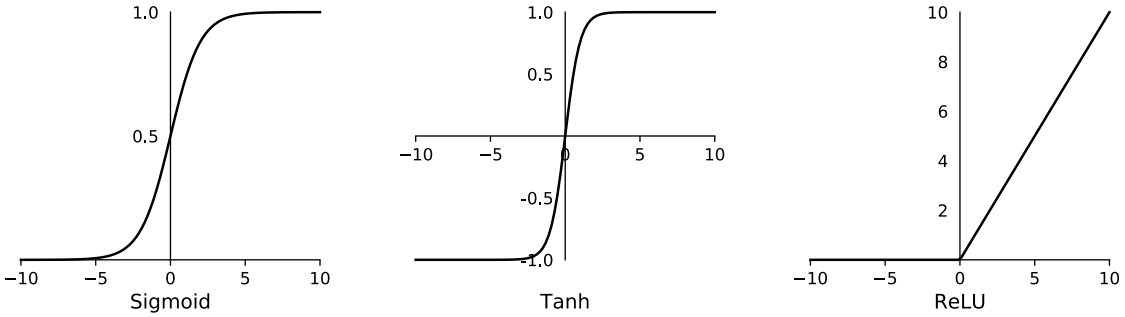


Figure 2.6.: Common activation functions: Sigmoid, Tanh, and ReLU

An optimized set of weight coefficients is learned throughout the learning process (i.e., training) of the ANN by minimizing a cost function (i.e., loss/error function). The cost function $E(w)$ can for example be the Sum of Squared Errors (SSE) between the target value and the predicted output

$$E(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \tag{2.15}$$

$z^{(i)}$ is the ith net input and $y^{(i)}$ the ith target value in the training set.

The optimization algorithm gradient descent is used to determine the weights which minimize the convex cost function. Gradient descent iteratively updates the weights in the opposite direction of the gradient $\nabla E(w)$ with a step-size of a defined learning rate η:

$$\Delta w_j = -\eta \frac{\delta E}{\delta w_j} = -\eta \sum_i (y^{(i)} - \phi(z^{(i)})) x_j^{(i)} \tag{2.16}$$

where w_j depicts the jth weight.

Interestingly, even though the first implementation of neural networks was in the 1950s, over the decades researchers began to lose interest in neural networks[15] (Rosenblatt, 1958). One of reasons was that there was no good solution for training networks with multiple layers until the backpropagation algorithm was popularized (Werbos, 1974; Rumelhart et al., 1986). Thereafter, in the past two decades, research done on multilayer neural networks (i.e., deep ANNs) has been vast. Figure 2.7 illustrates how multiple single neurons can be connected to a multilayer feedforward neural network (i.e., Multilayer Perceptron (MLP), Rosenblatt, 1961) with multiple so-called hidden layers between the input and the output layer.

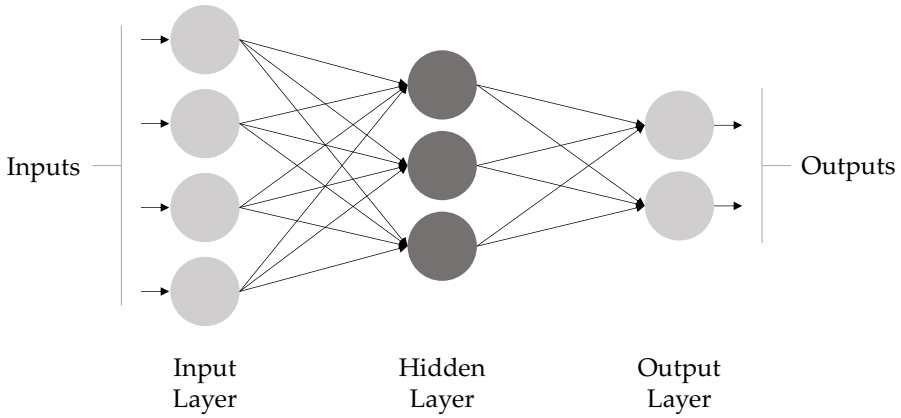


Figure 2.7.: MLP with one hidden layer (cf. Djuriš et al., 2012)

Generally, neural networks can be used for both, regression and classification tasks and have been applied to countless scientific and industrial problems, such as atmospheric sciences (cf. Gardner and Dorling, 1998), medical sciences (cf. Yan et al., 2006), computer vision (cf. Simonyan and Zisserman, 2015), or speech recognition (cf. Graves et al., 2013). In the following paragraph, we examine a specific architecture of neural networks, RNNs, which yields promising results in time series forecasting.

Recurrent Neural Networks

Traditional ANNs have the major shortcoming that they cannot incorporate information from previous events in the data when making predictions based on certain inputs. In

[15] Sometimes referred to as the AI winter, `https://en.wikipedia.org/wiki/AI_winter`, accessed on March 28, 2018

other words, as in classical ML, we assume that input data are completely independent of each other. However, when dealing with sequential data such as time series, spoken/written text, or video frames, we require to model these dependencies among inputs. RNNs, which were invented in the 1980s, can be interpreted as networks with memory which capture information from previous inputs over long or short term (Hopfield, 1982). In figure 2.8, a RNN is represented by its input x_t, a hidden layer A, and an output h_t[16]. The hidden layer cell (i.e., neuron) A has two outputs, one to the output layer and one feeding the output back into A through a loop which allows to persist information. In order to better understand how information is passed the loop is unrolled which reveals the chain-like nature of RNNs.

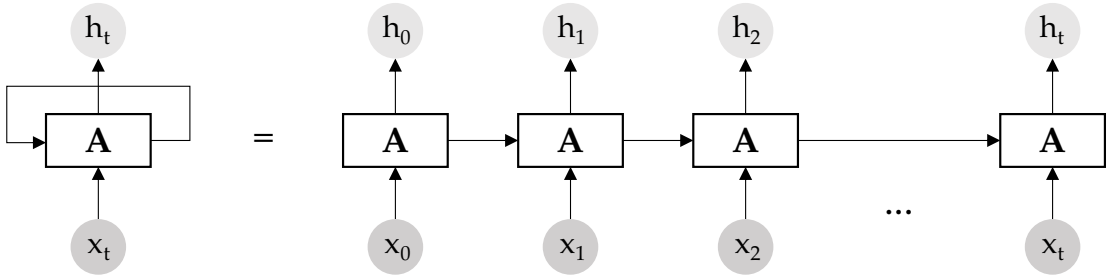


Figure 2.8.: Schematic of an unrolled RNN

Auspicious results have been achieved by applying RNNs to problems in speech recognition, language modeling, translation, image captioning[17]. At the core, the most successful form of RNNs are so-called Long Short Term Memory (LSTM) networks which are capable of learning long- and short-term dependencies among inputs. They extend the basic idea of RNNs by adding information flows and gates inside the hidden memory cells (i.e., LSTM units) to protect and control the cell state (Hochreiter and Schmidhuber, 1997). Figure 2.9 shows the information flows and gates inside LSTM units[18]. These include the forget gate which is responsible for only keeping certain information from the inputs x_t and h_{t-1}, the input gate which decides what information will be stored in the cell state C_t, and the output gate to filter what is the output h_t of the cell.

Cho et al. (2014) suggest the Gated Recurrent Unit (GRU), a variation of the LSTM unit, where forget and input gates are combined and cell state and hidden state are merged resulting in a simpler model which is becoming increasingly popular.

[16]Image from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, accessed on February 13, 2018

[17]See `http://karpathy.github.io/2015/05/21/rnn-effectiveness/` for an impressive collection of applications of RNNs, accessed on March 28, 2018

[18]Image from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, accessed on February 13, 2018

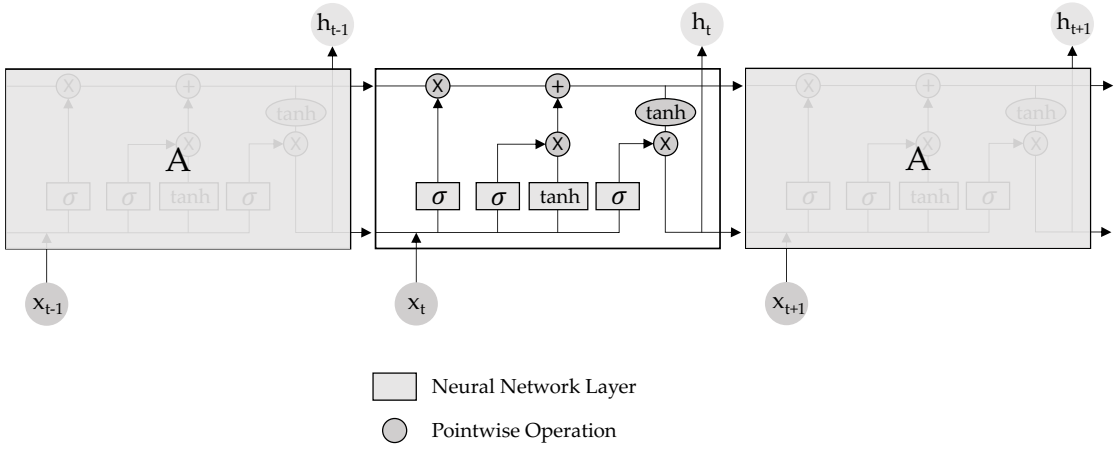


Figure 2.9.: Information flows and gates inside LSTM units

2.4. Anomaly Detection

In the context of data analysis, anomalies can be defined as patterns that do not reflect conformity compared to an observed normal behavior in the data. Often anomalies translate to critical actionable information, e.g., anomalous credit card transactions may indicate credit card or identify theft (Aleskerov et al., 1997). Hence, investigating these patterns is of particular interest for a large variety of application domains which constitutes the research area of anomaly detection. Depending on the area of application, non-conforming patterns have been referred to as anomalies, outliers, exceptions, or surprises (Chandola et al., 2009). Regardless of the terminology used, extensive research has been done on detection techniques in computer network security (i.e., intrusion detection) (cf. Kumar, 2005), fraud detection (cf. Aleskerov et al., 1997), health care (cf. Spence et al., 2001), or aerospace engineering (cf. Fujimaki et al., 2005).

Anomaly detection requires the definition of data representing normal behavior which then allows the identification of non-conforming, abnormal data points. Figure 2.10 illustrates two normal regions in the two-dimensional space, N_1 and N_2, which include the majority of observations from the data set. o_1, o_2, and O_3 are data points sufficiently distant from these regions and therefore can be interpreted as anomalies.

Based on Chandola et al. (2009) and Hodge and Austin (2004), we identify several challenges when designing anomaly detection techniques:

- **Blurred Boundaries Between Anomalous and Normal Regions:** This makes it difficult to distinguish between anomalous and normal observations which lie close to the boundaries.
- **Evolving Normal Behavior:** Future representations of normal behavior might be significantly different to the current notion.

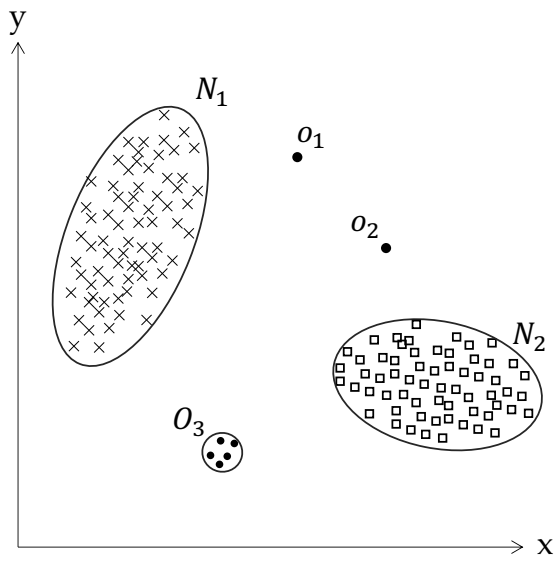


Figure 2.10.: Anomalies in a two-dimensional data set (Chandola et al., 2009)

- **Transferability of Techniques:** It is difficult to transfer assumptions about the notion of an anomaly to other application domains.
- **Availability of Ground Truth:** Often sparse availability of labeled normal and anomalous data.
- **Data Quality:** Data could contain noise which is not easily distinguishable from anomalies.
- **Nature of Input Data:** Both, high dimensional (e.g., multivariate) and heterogeneous (e.g., categorical, continuous) data can be challenging aspects for detection algorithms.
- **Type of Anomaly:** Different techniques might be best suitable for detecting point anomalies, contextual anomalies, or collective anomalies.
- **Type of Output:** Depending on the application domain the output of the anomaly detection technique should be a label, score, or confidence interval.

Extensive research in ML and statistical domains is done to meet specific requirements of various application domains (Agyemang et al., 2006; Chandola et al., 2009; Hodge and Austin, 2004; Patcha and Park, 2007). Accordingly, in addition to the separation based on availability of labels (i.e., unsupervised and supervised learning algorithms), the majority of techniques can be grouped into classification based, clustering based, nearest neighbor based, and statistical techniques. We mainly focus on applying clustering and nearest neighbor based anomaly detection techniques.

Clustering or cluster analysis separates data into groups where observations of the same group or cluster are in a way more similar than data points in other clusters. As this (usually) unsupervised way of learning structures in the unlabeled data can potentially reveal regions of normal and anomalous behavior, several anomaly detection techniques based on clustering have been developed. Chandola et al. (2009) categorize them based on their underlying assumptions regarding anomalies, where anomalies either (i) "do not belong to any cluster", (ii) "lie [...] far away from their closest cluster centroid", or (iii) "belong to small or sparse clusters" (Chandola et al., 2009). However, many of these techniques are not optimized for anomaly detection as they detect anomalies only as a by-product of the clustering itself. Furthermore, computational complexity and performance are highly dependent on the algorithm and chosen parameters, in case of a parametric clustering algorithm. Nevertheless, we apply Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996), a density-based clustering for anomaly detection and mainly consider assumption (i) and (ii) to decide whether observations are anomalies or not.

Additionally, Local Outlier Factor (LOF) (Breunig et al., 2000), an unsupervised learning technique to detect outliers based on the density of nearest neighborhoods, is applied.

2.4.1. DBSCAN

DBSCAN is designed to detect clusters of arbitrary shape with minimal domain knowledge required to determine the input parameters. The parameters of the algorithm are Eps, where the "Eps-neighborhood of a point p", $N_{Eps}(p)$, is defined as $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$, and $MinPts$, the minimum amount of data points within the Eps-distant neighborhood around p (Ester et al., 1996). Hence, clusters contain points which have a minimum of $MinPts$ neighbors within a neighborhood of radius Eps, i.e., the density of their neighborhood is higher than $MinPts$. Although this might pertain for core points of clusters, border points have fewer neighbors in their neighborhood, but should still belong to the respective cluster. Therefore, we require new definitions to determine the degree of connectivity among points belonging to clusters (Ester et al., 1996):

- **Directly Density-Reachable:** If a point p satisfies the conditions (i) $p \in N_{Eps}(q)$ and (ii) $|N_{Eps}(q)| \geq MinPts$, it is directly density-reachable from a point q.
- **Density-reachable:** If there is a chain of points $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = q$ such that each point p_{i+1} is directly density-reachable from p_i, a point p is density-reachable from q.
- **Density-Connected:** If two points p and q are both density-reachable from a point o, they are considered as density-connected.

- **Cluster:** If D is a set of data points, a cluster C is a non-empty subset of D which fulfills the conditions (i) $\forall p, q:$ if $p \in C$ and q is density-reachable from p, then $q \in C$, and (ii) $\forall p, q \in C : p$ is density-connected to q.
- **Noise:** Noise are all points that do not belong to any cluster C_i with $i = 1, \ldots, k$.

An abstract version of the DBSCAN algorithm is outlined in algorithm 2.1 [19].

Algorithm 2.1: Abstract DBSCAN Algorithm (Schubert et al., 2017)

```
Compute neighbors of each point to identify core points
Join neighboring core points into clusters
foreach non−core point do
    if core point in neighborhood (border point)
        add to cluster of core point
    else (noise point)
        add to noise
```

The complete pseudocode for the original DBSCAN algorithm is provided in algorithm 2.2. The RangeQuery(DB,dist,p,eps) function returns $N_{Eps}(p)$, the Eps-neighborhood of p, based on the chosen distance metric. Even though DBSCAN has never been constrained to the euclidean distance, it is the most commonly used distance metric (Ester et al., 1996; Sander et al., 1998).

Algorithm 2.2: Basic DBSCAN Algorithm (Ester et al., 1996; Schubert et al., 2017)

```
input: D: Data set
input: eps: Neighborhood−radius
input: minPts: Density threshold
input: dist: Distance metric (\eg euclidean)
output: label: Point labels, initially undefined
foreach point p in D do
    if label(p) ≠ undefined then continue
    Neighbors N ← RANGEQUERY(D, dist, p, eps)
    if |N| < minPts then
        label(p) ← Noise
        continue
    c ← next cluster label
    label(p) ← c
    Seed set S ← N \ {p}
    foreach q in S do
        if label(q) = Noise then label(q) ← c
        if label(q) ≠ undefined then continue
        Neighbors N ← RANGEQUERY(D, dist, q, eps)
        label(q) ← c
        if |N| < minPts then continue
        S ← S ∪ N
```

[19] `https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/` provides a great interactive visualization of the DBSCAN algorithm, accessed on March 28, 2018

Regarding the challenge of the parametrization of the algorithm, Ester et al. (1996) suggest a heuristic to setting the global parameters Eps and $MinPts$. More recently, Campello et al. (2013) have developed Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN), a hierarchical DBSCAN variant which performs DBSCAN over different Eps values and finds the clustering results which yield the best stability over Eps and thus is more robust to parameter selection (Campello et al., 2013; McInnes et al., 2017).

2.4.2. LOF

LOF is an outlier detection technique proposed by Breunig et al. (2000). Different to other approaches for outlier detection, LOF outputs "a degree of being an outlier" of an observation rather than a binary property, i.e., outlier or not (Breunig et al., 2000). The approach of LOF is loosely related to the concept of density-reachability of DBSCAN, but unlike DBSCAN is not optimized to find clusters, but to detect outliers. Furthermore, LOF emphasizes to incorporate how isolated an observation is with respect to the density of the local surrounding neighborhood. In alignment with Breunig et al. (2000), we outline the formal definitions behind LOF:

- **k-Distance:** The k-distance of an observation p, is defined as the distance $d(p,o)$ between p and an observation $o \in D$ with (i) at least k observations $o' \in D \setminus \{p\}$ where $d(p,o') \leq d(p,o)$, and (ii) at most k-1 observations $o' \in D \setminus \{p\}$ where $d(p,o') < d(p,o)$.
- **k-Distance Neighborhood:** Based on the k-distance of an observation p, the k-distance neighborhood of p $N_k(p)$ is defined as $N_k(p) = \{q \in D \setminus \{p\} | d(p,q) \leq k\text{-}distance(p)\}$. The observations q are called the k-nearest neighbors of p.
- **Reachability Distance:** The reachability distance of an observation p with respect to observation o is defined as $reach\text{-}dist_k(p,o) = max\{k\text{-}distance(o), d(p,o)\}$. Hence, statistical fluctuations of $d(p,o)$ can be significantly reduced for all p close to o.
- **Local Reachability Density:** The local reachability density of an observation p is defined as $lrd_{MinPts}(p) = 1/(\frac{\sum_{o \in N_{MinPts}(p)} reach\text{-}dist_{MinPts}(p,o)}{|N_{MinPts}(p)|})$ where $MinPts$ is the same parameter as in DBSCAN.
- **(Local) Outlier Factor:** The (local) outlier factor of an observation p can be calculated by the formula:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|} \tag{2.17}$$

  Hence, the outlier factor is the "average of the ratio of the local reachability density of p and those of p's $MinPts$-nearest neighbors" (Breunig et al., 2000). The lower

p's local reachability density and the higher the local reachability densities of p's neighborhood, the higher is degree of being an outlier of p.

In conclusion, the outputted factor (LOF) is a numerical value which reflects the degree of outlier-ness and can thus be interpreted as an anomaly score of an observation.

The discussed anomaly detection techniques complete the fundamental terminologies which are relevant and methodologies that have been applied within this thesis. The next chapter gives an overview about the reviewed related work from previous research done in the domain of APM with respect to time series analysis and anomaly detection.

3. Related Work

In this chapter, the prevalent scientific work done in related research areas is reviewed. Section 3.1 describes the approach of the review which has been defined in alignment with Webster and Watson (2002). Afterwards the findings of the literature review are presented and put into context to answer RQ_1 and $RQ_{2.1}$ and to specify the RCs.

Recapitulation RQ_1

What are research gaps in existing scientific work regarding enterprise application performance forecasting and detection of abnormal system behavior in enterprise applications?

Recapitulation $RQ_{2.1}$

Which are relevant APM measures to assess enterprise application performance?

3.1. Literature Review Approach

The following approach is based on advice from Webster and Watson (2002) who propose a process for literature reviews, which includes (i) identifying relevant literature (see subsection 3.1.1), (ii) structuring the review (see subsection 3.1.2), and (iii) closing the research gap (see subsection 3.1.3) by developing your own contributions. The final result is a set of identified related scientific work which helps us to define the scope of this thesis.

3.1.1. Identification of Relevant Literature

To identify the relevant literature, we use the online search catalogue Scopus[1] which as of February 2018 is the largest abstract and citation database for scientific journals, books, and conference proceedings. In accordance with RQ_1, we define two Search Queries (SQs) incorporating the two sided research focus on performance forecasting and anomaly detection in APM:

[1] `https://www.scopus.com`, accessed on February 28, 2018

- **SQ_1:** TITLE-ABS-KEY(("Application Performance Management" OR "Application Performance Monitoring" OR "Server Performance" OR "Server Monitoring") AND ("forecast" OR "machine learning" OR "time series" OR "prediction" OR "data mining"))

- **SQ_2:** TITLE-ABS-KEY(("Application Performance Management" OR "Application Performance Monitoring" OR "Server Performance" OR "Server Monitoring") AND ("outliers" OR "anomalies" OR "anomaly detection" OR "abnormal"))

Since in scientific work APM is often related to the field of (web) server performance, the queries are formulated accordingly. Furthermore, keywords describing methodologies need to be considered as well, e.g., ML, data mining, or anomaly detection. SQ_1 combines APM and performance forecasting which addresses $RQ_{2.1}$ and $RQ_{2.2}$. SQ_2 refers to RQ_3 and thus integrates APM with anomaly detection. Generally, the queries are used to perform a TITLE-ABS-KEY search, which means title, abstract, and keywords of scientific sources are examined. The search results of SQ_1, SQ_2, and their combination (i.e., intersection) are displayed in table 3.1.

Table 3.1.: Complete search results from Scopus, February 28, 2018

SQ	Result Count
SQ_1	76
SQ_2	19
SQ_1 AND SQ_2	3

Certainly only a subset of the initial search results are actually related work to the topic this thesis. Hence, we need to filter out those articles which are not related to the research areas we are interested in. Irrelevant results from SQ_1 comprise work on biology (e.g., protein structures), specific web server profile modelling (e.g., user session modelling), and infrastructure/platform-specific research (e.g., configuration of proxy servers, investigation of Java Message Services message throughput bottlenecks, or wireless network environments). Results from SQ_2 include work on traffic overload analysis in sensor networks, user behavior modeling, and anomaly detection techniques which are not of particular relevance. Table 3.2 gives an overview of the filtered search results retrieved from Scopus. After filtering we are still left with thirty-eight articles for SQ_1 and eight articles for SQ_2 of which three are part of both result sets.

In the subsection 3.1.2, the review is organized into concepts whereas in subsection 3.1.3 the subset of the most relevant related work is further narrowed down to identify research gaps.

Table 3.2.: Filtered relevant search results from Scopus

SQ	Result Count
SQ_1	38
SQ_2	8
SQ_1 AND SQ_2	3

3.1.2. Structuring the Review

Following the guideline from Webster and Watson (2002), we structure the review by concepts. The concept-centric approach organizes the review as we compile a concept matrix to group key concepts discussed in related literature. Based on the filtered search results, we identify eight core concepts, namely Web Server Performance (C1), APM (C2), Performance Forecasting (C3), Resource Prediction (C4), Modelling/Profiling (C5), Statistical Modeling (C6), Data Mining/ML (C7), and Anomaly Detection (C8). Each article can discuss multiple concepts. However, all filtered articles are at least required to discuss either C1 or C2. The total amounts of filtered search results discussing each concept are listed in table 3.3.

Table 3.3.: Concept matrix with eight core concepts

SQ	C1	C2	C3	C4	C5	C6	C7	C8
SQ_1	24	15	10	4	13	8	11	3
SQ_2	4	4	0	1	1	1	3	7
SQ_1 AND SQ_2	2	1	0	0	0	1	2	3

3.1.3. Identification of Research Gap

As outlined in section 1.2, we are particularly interested in work related to APM as well as performance forecasting, data mining/ML, and anomaly detection. Therefore, we further investigate articles discussing the concepts C1 or C2, and C3, C7, or C8.

Performance Forecasting ((C1 OR C2) AND C3)

SQ_1 reveals ten articles discussing performance forecasting regarding different aspects. Hoffmann et al. (2007) provide a best practice guide for predicting system performance variables which they derived from analysis done on the Apache web server[2]. Their work covers short-term (5 minutes ahead) and long-term (48 hours) prediction of the server

[2] `https://httpd.apache.org/`, accessed on February 28, 2018

response time and free physical memory by using linear regression models and models with nonlinear Support Vector Machines (SVMs). Borzemski and Kamińska-Chuchmała, in a series of publications between 2011 and 2016, propose the use of the turning bands method, a statistical modelling approach, and spatial regression models to predict web server performance (Borzemski and Kamińska-Chuchmała, 2011; 2011; 2013; 2014; 2016). As key performance indicator they define the response time to download a complete Hypertext Markup Language (HTML) page from the web server. Amani et al. (2011) use a nonlinear autoregressive neural network model with exogenous inputs to do multi-step ahead response time predictions for database queries. Xu et al. (2010) apply Principal Component Analysis (PCA) and multi-dimensional time series analysis to predict performance regression of the software (i.e., application server aging) in an experimental setting. However, they focus on single-step time series forecasting of response time and throughput with linear models. Contrary to the latter, Peng et al. (2017) use nonlinear RNNs (see subsection 2.3.2) to predict current web server error rate, throughput and response time from log data. Last, Panwar (2013) emphasizes a shift towards proactive APM techniques rather than implementing reactive monitoring.

After considering keywords and citations of identified articles in accordance to Webster and Watson (2002), the SQ is slightly modified to incorporate scientific work specifically done on forecasting the response time of computer systems:

- **$SQ_{1.1}$:** TITLE-ABS-KEY(("Response time (computer systems)") AND ("forecast" OR "machine learning" OR "time series" OR "prediction" OR "data mining"))

Searching for $SQ_{1.1}$ leaves us with 641 search results. Due to the relatively widespread field of response time (e.g., computer engineering, system architecture, database design, memory/cache management) most of the results are irrelevant for our purpose. After rigorous filtering we find another three articles which are worth to be mentioned: Albu et al. (2013) have compared twenty-eight neural network architectures to predict the response time of cloud-based web services. Their best estimator is a MLP (see subsection 2.3.2). Yanggratoke et al. (2012) predict the response times of a distributed key-value storage system within the Spotify[3] backend through a statistical model of the request load. Zhang et al. (2014) forecast website response time with SVMs on an hourly basis.

Based on the outlined related work, we identify the research gap with respect to performance forecasting in the comparison of linear and nonlinear ML techniques (e.g., tree-based regressors, neural networks). Even though work has been done on both of these areas separately, these methodologies have not yet been applied to the exact same data set. Furthermore, only little research has been done on multi-step and multivariate performance forecasting. Hence, we aim to define a generic multi-step multivariate time series forecasting problem based on theoretically arbitrarily available monitoring measures (e.g., from APM software such as Dynatrace, see subsection 2.1.2). This is

[3] `https://www.spotify.com`, accessed on February 28, 2018

different to most prior research which focuses on specific platforms and data sources, e.g., log data from web servers such as Apache web server or NGINX[4]. The key APM measure indicating enterprise application performance we attempt to forecast is defined in section 3.2 where $RQ_{2.1}$ is answered.

Data Mining/Machine Learning ((C1 OR C2) AND C7)

Since three articles have already been discussed in the previous subsection and will not be considered again, SQ_1 yields another eight search results for data mining/ML related concepts in the domain of APM or server monitoring. Web Usage Mining (WUM) – discussed by Baraglia and Palmerini (2002), Hussain et al. (2010), and Feng and Vij (2007) – uses clustering or classification algorithms to classify web visitors, generate suggested links for user navigation, or perform web cache pre-fetching which can potentially improve web server performance by managing the web resources effectively. Based on multiple server monitoring time series collected at Yahoo[5], Hyndman et al. (2016) first perform a PCA to reduce the dimensionality of the multivariate time series. Afterwards they apply a density-based and an α-hull based multi-dimensional outlier detection algorithm (Hyndman, 1996). Jiang et al. (2005) perform regression analysis on performance data in order to discover correlations among workload and various system parameters in Oracle[6] database systems and Microsoft Internet Information Services[7]. Wang et al. (2016) propose an Auto-Regressive Integrated Moving Average with Back-Propagation (ARIMA-BP) neural network to forecast CPU utilization based on time series on resource utilization. A rather irrelevant approach for the given context is discussed by Ohta (2016) who derives a mapping function from passive traffic monitoring and electric currents to the actual server performance by using decision trees. Finally, Cunha and Moura E Silva (2013) discover that bayesian networks are superior to SVMs and decision trees in predicting and classifying failures in video-streaming services on web servers.

SQ_2 reveals another three articles addressing C7, but only one is not discussed yet: Cunha and Moura E Silva (2012) use decision trees and naive bayes to classify and separate performance anomalies from workload-explained failures in video-streaming web servers.

There are various of limitations in the outlined prior research. A variety of ML techniques (e.g., ensemble methods and RNNs), especially those suitable for regression problems, have not yet been evaluated regarding the task of predicting performance of enterprise applications. Furthermore, only few articles are identified which perform unsupervised ML to detect performance anomalies. This is further investigated in the next paragraph.

[4] `https://www.nginx.com`, accessed on Februray 28, 2018
[5] `https://www.yahoo.com`, accessed on February 28, 2018
[6] `https://www.oracle.com`, accessed on February 28, 2018
[7] `https://www.iis.net/`, accessed on February 28, 2018

Anomaly Detection ((C1 OR C2) AND C8)

There are three articles identified by SQ_1 and four additional ones by SQ_2. Excluding the already discussed ones, we are left with four further articles. Ahmed et al. (2016) provide an overview and a comparison regarding ease of installation and approaches for detecting performance regressions or anomalies of a variety of APM software solutions (see subsection 2.1.2). Ara et al. (2017) propose a bivariate time series model for statistical monitoring of a web server for detecting abnormal error rates. To reveal implementation issues of different web servers, Takashi et al. (2006) analyze performance anomalies in network delay and packet loss ratio. Last, Guo et al. (2011) use modified SVMs to detect abnormal patterns in server performance monitoring data to define respective thresholds for alarm systems.

The area of anomaly detection in APM/server performance offers opportunities for a variety of research fields. Among the identified related work only Hyndman et al. (2016) use bivariate outlier detection techniques to detect anomalies. Hence, we aim to propose an approach of applying density-based unsupervised learning algorithms for anomaly detection which takes generic multivariate monitoring data (i.e., time series) as an input and provides context information on which variable (i.e., monitoring measure) an anomaly was suspected.

3.2. Findings from Literature Review

From the reflections on research gaps in the previous section, we constitute RC_1 and RC_2 to meet limitations in prior research as follows:

- **RC_1:** Implementation and comparison of the linear and nonlinear ML techniques for multivariate multi-step time series application performance forecasting: linear regression, tree-based regressors, and neural networks.
- **RC_2:** Implementation of a multivariate density-based anomaly detection model providing an indication for the root cause of abnormal system behavior.

Furthermore, to answer $RQ_{2.1}$, we outline the relevant APM measures to assess enterprise application (i.e., server) performance based on the identified related work about performance forecasting. By revisiting the ten articles discussing concept C3, the key APM measures indicating enterprise application performance listed in table 3.4 are found.

The analysis confirms the assumptions suggested in subsection 2.1.3 about response time being the most relevant APM measure for assessing performance of enterprise applications.

Table 3.4.: Key APM measures to assess enterprise application performance from identified literature

APM Measure	Count	Sources
Response Time	**9**	Amani et al., 2011; Hoffmann et al., 2007 Borzemski and Kamińska-Chuchmała 2011; 2011; 2013; 2014; 2016 Xu et al., 2010; Peng et al., 2017
Throughput	2	Xu et al., 2010; Peng et al., 2017
Error Rate	1	Peng et al., 2017
Free Physical Memory	1	Hoffmann et al., 2007

To conclude the literature review, we were able to identify gaps in the research done on APM in combination with performance forecasting and anomaly detection. These are addressed by two RCs, RC_1 and RC_2 to fill these gaps and thus answer RQ_1. Furthermore, from the related scientific work we were able to answer $RQ_{2.1}$ and selecting response time as the most relevant APM measure. These insights are crucial as we continue to elaborate on $RQ_{2.2}$ and the forecasting of enterprise application performance respectively.

Abridged Answer to RQ_1

The identified research gaps are:

- **Enterprise Application Performance Forecasting:** Investigation of **linear and nonlinear** ML techniques for **multivariate multi-step** time series performance forecasting.
- **Detection of Abnormal System Behavior:** Investigation of **multivariate density-based anomaly detection** techniques providing an **indication for the root cause** of abnormal system behavior.

Abridged Answer to $RQ_{2.1}$

From the reviewed related scientific (and non-scientific, see subsection 2.1.3) work, **response time** can be identified as the **most relevant APM measure** which reflects the performance of an enterprise application.

4. Predicting Enterprise Application Performance

In this chapter, we elaborate on $RQ_{2.2}$ and develop RC_1, an implementation and comparison of linear and nonlinear ML models for enterprise application performance forecasting. By reviewing related scientific work in chapter 3, we identify response time as the key APM measure indicating performance. Hence, the models are evaluated regarding their predictive quality for response time and $RQ_{2.2}$ is specified for this purpose.

Recapitulation $RQ_{2.2}$

How well can the response time of an enterprise application be predicted based on historical APM measures?

Recapitulation RC_1

Implementation and comparison of the linear and nonlinear ML techniques for multivariate multi-step time series application performance forecasting: linear regression, tree-based regressors, and neural networks.

4.1. Define Objectives

The defined scope for the RC_1 implicates requirements (depicted by Req_i in the following) regarding the development of the first research artifact, i.e., a forecasting model for response time in enterprise applications.

- **Req_1:** The model suits a multi-step output time series forecasting problem.
- **Req_2:** The model can take both, univariate and multivariate feature vectors as input to incorporate multiple APM measures.
- **Req_3:** The prediction of the model is comparable regarding different evaluation metrics on the same data subset.

The goal of this chapter is to compare linear regression, random forest regressor, MLP, and LSTM models, which meet these requirements regarding their predictive quality. In the end, the research artifact is declared as the best performing model.

4.2. Design & Development

This section encompasses the design and development process of the performance forecasting models. The data used for the analyses and predictive modeling is obtained from the APM systems of two enterprise applications at a German car manufacturer. The applications are selected based on their high business relevance, and the quality as well as the availability of monitoring data. In the following, they are depicted by App_1 and App_2. Together with experts from a specialized department for APM and those responsible for the development and operation of respective applications, the scope of the analyses is defined as follows:

- **Time Period of Collected Monitoring Data:** Six weeks, November 6, 2017 to December 17, 2017
- **Input:** Fifteen APM measures with time series data of varying collection intervals, from 1 data point per minute to 1 data point per 60 minutes (step size defined as 1 minute)
- **Output:** Predictions of Average Response Time (ART) for 30 minutes in $30x1$ minute steps

Subsection 4.2.1 provides a thorough overview of the data set that is used. In order to keep track of the state of the data through the different preparation and transformation steps, an overview of a data frame containing exemplary data is provided at each of the subsequent subsections. Before the actual predictive modeling, the unprocessed data frame traverses through the three states described in figure 4.1.

Step	Description	Section
1	Unprocessed data frame after loading data from monitoring sources	4.2.1
2	Cleaned and pre-processed data frame without missing values and framed as a multi-step time series forecasting supervised learning problem	4.2.2
3	Data frame containing a feature vector of (self-)engineered and selected features	4.2.3

Figure 4.1.: Description of three data frame processing steps throughout section 4.2

4.2.1. Data Source Identification

The available data sources for monitoring data of App_1 and App_2 are the APM software Nagios, JMX, and Dynatrace (see subsection 2.1.2). The typical three-layer enterprise application architecture is extended by an intermediate web server layer, as web server traffic is monitored separately (Fowler, 2002). In figure 4.2, the APM measures collected by the monitoring systems across the four-layer architecture are outlined with their respective collection intervals. Since client data is not available, we can only consider server monitoring data from the point where a client request arrives at the web server layer where a load balancer sends the request to an application server instance. The available APM measures are as follows, note, however, that some of them are only available for App_2:

- **Requests per Minute:** The amount of requests from clients arriving at the web server layer. The measure is aggregated per minute, but the collection interval is approximately every 1.5 minutes.
- **Traffic per Minute**: The amount of data in bytes transferred through the web server layer. The measure is aggregated per minute, but the collection interval is approximately every 1.5 minutes.
- **CPU Load:** The average CPU utilization in percentage. The collection interval is approximately every 1.5 minutes.
- **MQS Queuelength:** The average length of the message queue. The collection interval is approximately every 2 minutes.
- **JVM Threads:** The average amount of JVM threads in the JVM running the application. The collection interval is approximately every 1.5 minutes.
- **JVM Heap Used:** The average heap memory used in the JVM running the application in megabytes. The collection interval is approximately every 1.5 minutes.
- **Response Time:** The response time of application servers in milliseconds. The response time is aggregated and collected as an average and a maximum measure. The collection interval is exactly 1 per minute.
- **Path Duration:** The complete duration a path was executed by the application server in milliseconds. Additionally to the response time, this measure can contain asynchronous subsequent tasks. The path duration is aggregated and collected as an average and a maximum measure. The collection interval is exactly 1 per minute.
- **Failed Transactions:** Amount of failed transactions includes every transaction resulting in an HTTP error. The collection interval is exactly 1 per minute.
- **Path Amount:** Amount of processed execution paths (i.e., traces). The collection interval is exactly 1 per minute.

- **Exception Count:** Amount of exceptions thrown by the application server. The collection interval is exactly 1 per minute.
- **Node Count:** Total amount of nodes (i.e., Dynatrace agents) visited in all processed paths. The collection interval is exactly 1 per minute.
- **DB Sessions:** Amount of current database sessions. The collection interval is approximately 1 every 60 minutes.
- **DB Physical Reads:** Amount of reading operations in database. The collection interval is approximately 1 every 60 minutes.
- **DB Physical Writes:** Amount of writing operations in database. The collection interval is approximately 1 every 60 minutes.

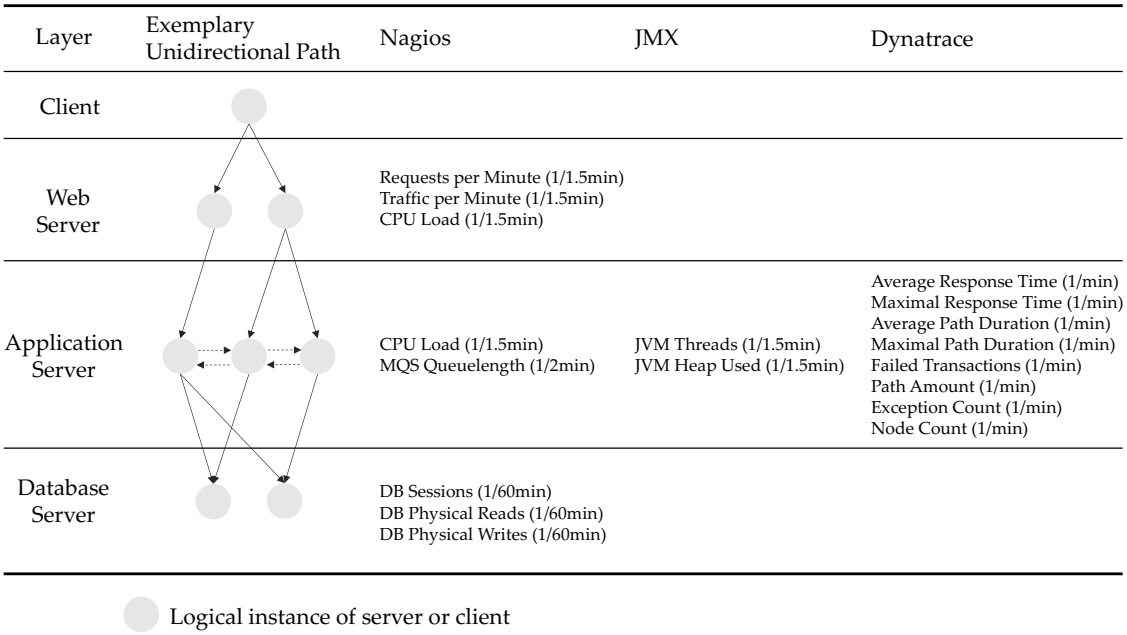


Figure 4.2.: Collected APM measures on four-layer architecture from three monitoring systems: Nagios, JMX, Dynatrace

The Nagios and JMX monitoring data is already provided in InfluxDB time series databases. However, Dynatrace produces path (i.e., execution trace) data which needs to be aggregated to time series format in advance. Figure 4.3 illustrates how data is extracted, aggregated into InfluxDB time series databases, and fetched via HTTP into Jupyter[1] notebooks[2] for conducting analyses.

[1] `http://jupyter.org/`, accessed on March 7, 2018

[2] Jupyter notebooks are documents which contain both, computer code, and rich text elements (paragraph, equations, figures, links, etc.), see `http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html`, accessed on March 7, 2018

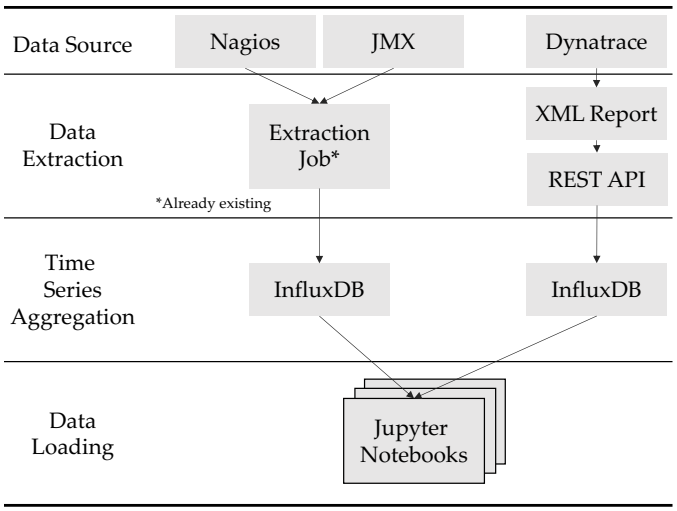


Figure 4.3.: Data extraction, aggregation and loading process

The process contains the following three steps:

1. **Data Extraction:** Within this step the data is extracted from the data source. For Nagios and JMX there is an existing extraction job which regularly fetches newly available monitoring data. Dynatrace offers an Application Programming Interface (API) which allows to query Extensible Markup Language (XML) reports. These reports aggregate path (i.e., execution trace) data on certain collection intervals. We use the minimum interval size of 1 data point per minute.
2. **Time Series Aggregation:** This step comprises storing an aggregated time series for each available enterprise application measure into instances of InfluxDB, a dedicated time series database.
3. **Data Loading:** The data loading is done via an exposed HTTP API of InfluxDB. The HTTP requests are sent from the Jupyter Notebooks which are the main tool we are using to conduct analyses.

The resulting data frame after the data loading is illustrated in figure 4.4.

4.2.2. Data Preparation

The dimensions of the data frame at this point are the fifteen outlined measures as columns and 60,480 rows for the six weeks of data (see figure 4.4).

Next, the data preparation encompasses the following three main steps:

- **Data Imputation:** Since the aforementioned available measures (i.e., time series) have different collection intervals, on a minutely basis many rows contain unknown

1 Unprocessed data frame after loading data from monitoring sources

Timestamp	Average Response Time	Exception Count	Traffic per Minute	Webserver CPU Load	DB Sessions	...
2017-11-06 00:00	669.45	1520	127.4	0.34	NaN	...
2017-11-06 00:01	301.23	1245	NaN	NaN	1133	...
2017-11-06 00:02	643.34	1993	115.3	0.26	NaN	...
...	...	...	...	...	...	...
2017-12-17 23:57	252.34	1024	95.4	2.51	NaN	...
2017-12-17 23:58	1034.25	1914	NaN	NaN	1342	...
2017-12-17 23:59	1144.28	2532	118.2	0.31	NaN	...

Figure 4.4.: Data frame after data loading

values (i.e., non-numeric (NaN) or null values). Data imputation describes the task of filling these missing gaps in the data with values. Among three tested filling approaches, namely mean, median, and forward filling, the median filling yields the best results regarding the final prediction. Hence, we fill missing values with the median value of the respective column (i.e., time series distribution).

- **Multi-step Forecasting Problem:** To meet Req_1, the time series are framed into a supervised learning problem with an output of 30 minutes as individual minutely time steps. We use equation 2.9 outlined in subsection 2.3.1, to create a multi-step forecasting problem with horizon h which is set to 30. The amount of prior time steps to consider (i.e., time lag, see subsection 2.3.1), often depicted by d, is initially also set to 30, meaning that the previous 30 minutes of each measure are used as the feature vector X. The exact composition of X is explained in subsection 4.2.3.
- **Split Data into Training and Testing Subsets:** Req_3 demands to define a subset of the data, which is used to compare different models regarding their predictive quality. Thus, we partition the initial data set into a train and a test data subset. Setting the split ratio is ambivalent, with less training data model parameters have greater variance, with less testing data performance has greater variance. Based on common practice in ML, we set the split ratio to 90:10, meaning the test data contains the last 10% of the total data set (Raschka, 2017).

After the data preparation the data frame looks as depicted in figure 4.5. The feature vector X is concretized in the next subsection.

4.2.3. Feature Engineering

In this subsection, we delineate the process of feature engineering and feature selection. These deliberations comply with Req_2 which postulates to consider both, univariate and multivariate time series problem modeling. In alignment with the Cross-Industry Standard Process for Data Mining (CRISP-DM), we review the features based on results

1 Unprocessed data frame after loading data from monitoring sources

Timestamp	Average Response Time	Exception Count	Traffic per Minute	Webserver CPU Load	DB Sessions	...
2017-11-06 00:00	669.45	1520	127.4	0.34	NaN	...
2017-11-06 00:01	301.23	1245	NaN	NaN	1133	...
2017-11-06 00:02	643.34	1993	115.3	0.26	NaN	...
...	...	...	...	...	...	...
2017-12-17 23:57	252.34	1024	95.4	2.51	NaN	...
2017-12-17 23:58	1034.25	1914	NaN	NaN	1342	...
2017-12-17 23:59	1144.28	2532	118.2	0.31	NaN	...

Target measure
30-minute forecast

2 Cleaned and pre-processed data frame without missing values and framed as a multi-step time series forecasting supervised learning problem

	ART(t-29)	...	ART(t)	EC(t-29)	...	EC(t)	...	ART(t+1)	...	ART(t+30)
90% Training Data	669.45	...	309.23	1520	...	2423	...	345.21	...	423.23
	301.23	...	451.34	1245	...	1133	...	425.26	...	542.76
	643.34	...	520.34	1993	...	2154	...	738.62	...	643.25
	...	...	...	...	...	...	...	...	...	...
	923.34	...	664.32	1842	...	1221	...	356.31	...	252.34
10% Test Data	1223.52	...	848.35	3728	...	1561	...	903.63	...	1034.25
	902.42	...	938.31	2114	...	1256	...	993.25	...	1144.28

X
(initial feature vector with lag = 30)

y
(output vector)

Figure 4.5.: Data frame after data preparation

from the modeling process (see subsection 2.3.1) and perform feature selection based on their importance for predictive accuracy (Shearer et al., 2000; Kohavi and John, 1997). The feature engineering and selection process is further specified in figure 4.6.

Whenever we generate a potential feature or feature set, we denote it with F_i. For App_1, we provide figures and plots to outline the feature engineering process (see appendix A.1.1 for App_2). The results are two sets of features, one for each application, containing the most important univariate and multivariate features for predicting subsequent 30 minutes.

Univariate Features

Univariate time series forecasting implies generating features from only one variable, in our case ART. In order to find useful features, we perform analyses on the historical

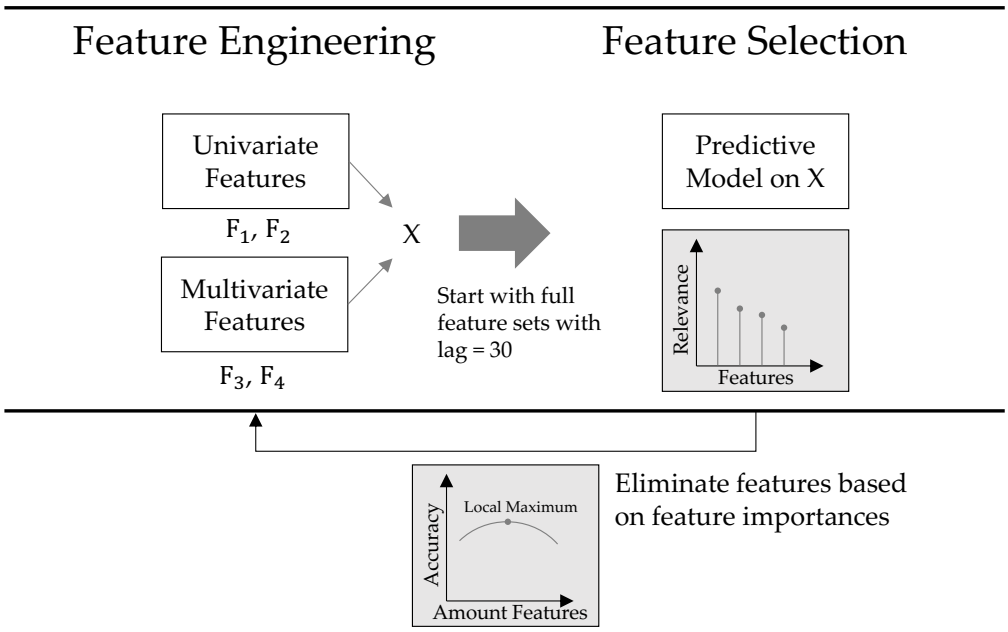


Figure 4.6.: Feature engineering and selection process

values of ART to detect cycles, trends, or seasonalities in the time series. A historical plot of the ART of App_1 with logarithmic y-scale is given in figure 4.7.

It is reasonable to consider the autocorrelation among time lags within the time series. Mathematically, the autocorrelation between to observation y_t and $y_{t+\tau}$ in a time series is defined as

$$R(\tau) = \frac{E[(y_t - \mu)(y_{t+\tau} - \mu)]}{\sigma^2} \tag{4.1}$$

where E is the expected value, μ the mean, and σ the variance of a time series. τ can be interpreted as the time lag (i.e., time steps in y) and R is a value between -1 and 1. Figure 4.8 illustrates these correlations up to a time lag of 1440 (i.e., one day in minutes).

The insights we gain from these analyses lead to the following feature sets:

- **F_1:** The autocorrelation plot clearly shows a declining correlation between ART observations over time. As we do not know which level of autocorrelation is a reasonable threshold to consider the respective time lags as potential features, we test a set of different lags. Initially, a lag d of 30 is specified and then increased up to 120. From iteratively assessing the prediction quality of the estimators defined in subsection 4.2.4, we find that 120 is the best value for d. The first set of features, F_1, is thus derived as
  - d prior time steps

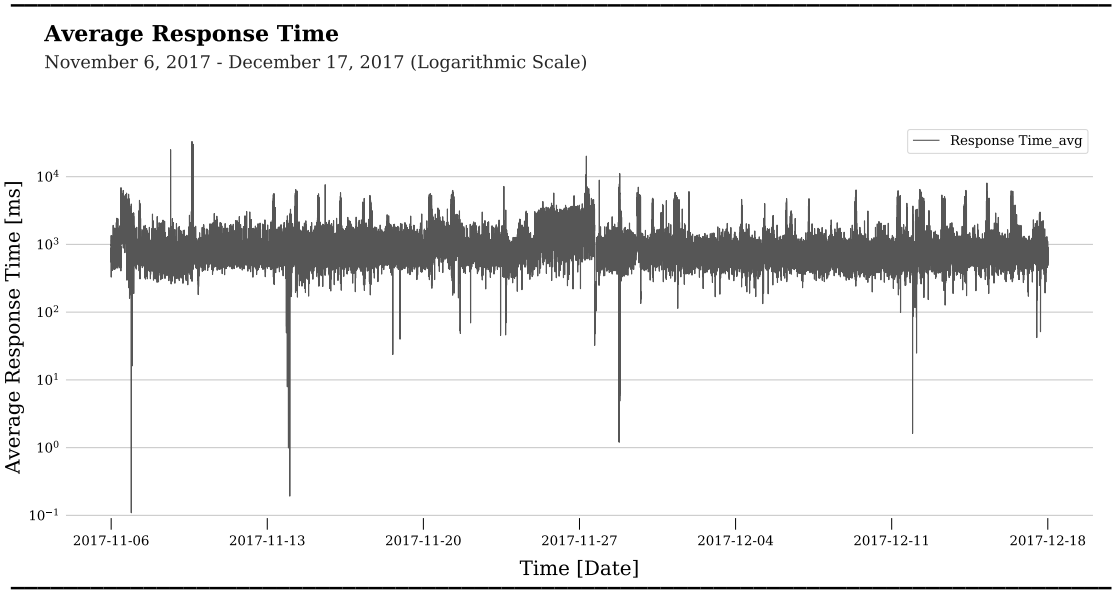


Figure 4.7.: Logarithmic plot of historical ART of App_1

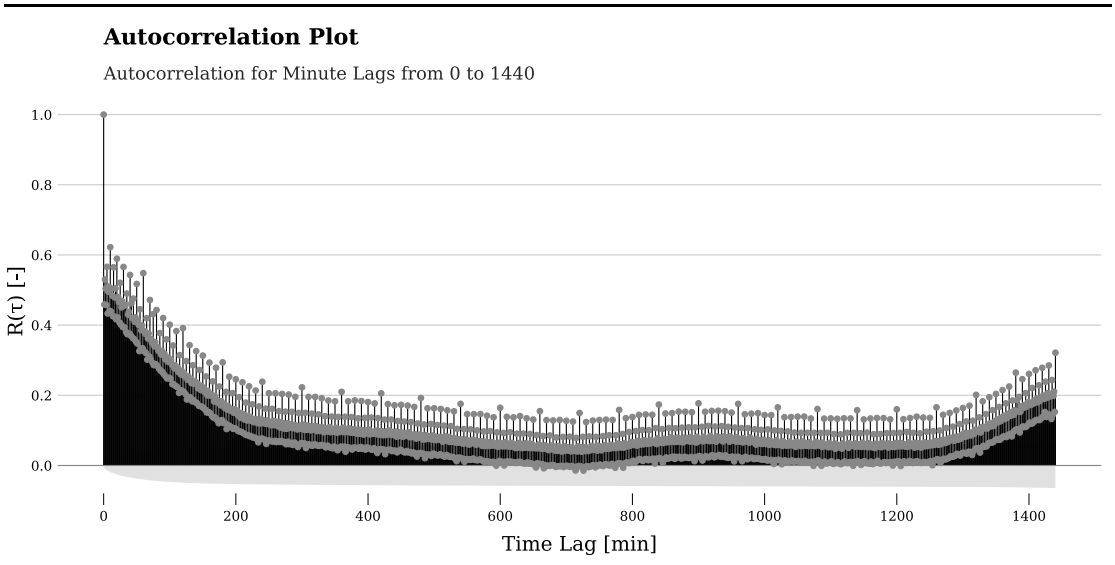


Figure 4.8.: Autocorrelation plot of historical ART of App_1

for the prediction of subsequent 30 minutes.

- **F_2:** The time series is stationary, connoting that mean and variance do not change over time. There are no trends or seasonalities that allow feature engineering. However, the autocorrelation analysis yields that for App_1 there is a slight daily cycle of ARTs. The second set of features, F_2, contains aggregation features on the previous hour and day and the combination of the latter:
  - Average ART of the last hour
  - Maximum ART of the last hour
  - Average ART of the last hour on the day before
  - Average ART of $+/-$ 30 minutes on the day before
  - Maximum ART of $+/-$ 30 minutes on the day before
  - Difference average ART hour before to average ART hour before on the day before

Multivariate Features

As aforementioned, there are fifteen APM measures which are extracted from different monitoring sources which can be used for multivariate feature engineering.

The following multivariate features are deduced:

- **F_3:** The first feature set is derived from domain knowledge. We self-engineer four features by combining monitoring measures and define F_3 as:
  - Average Failure Rate (i.e., ratio Failed Transactions to Path Amount)
  - Average Exceptions per Path
  - Average Failures per Path
  - Average Nodes per Path
- **F_4:** To get insights on the correlation of ART to the different measures, we perform a correlation analysis (see figure 4.9). The Pearson Correlation Coefficient (PCC) is used:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{4.2}$$

  We define $|\rho_{X,Y}| \geq 0.05$ as the threshold for considering measures as significantly correlated to ART. Consequently, F_4 contains the following features:
  - Exception Count
  - Maximal Response Time

- Average Path Duration
- Node Count
- Path Amount

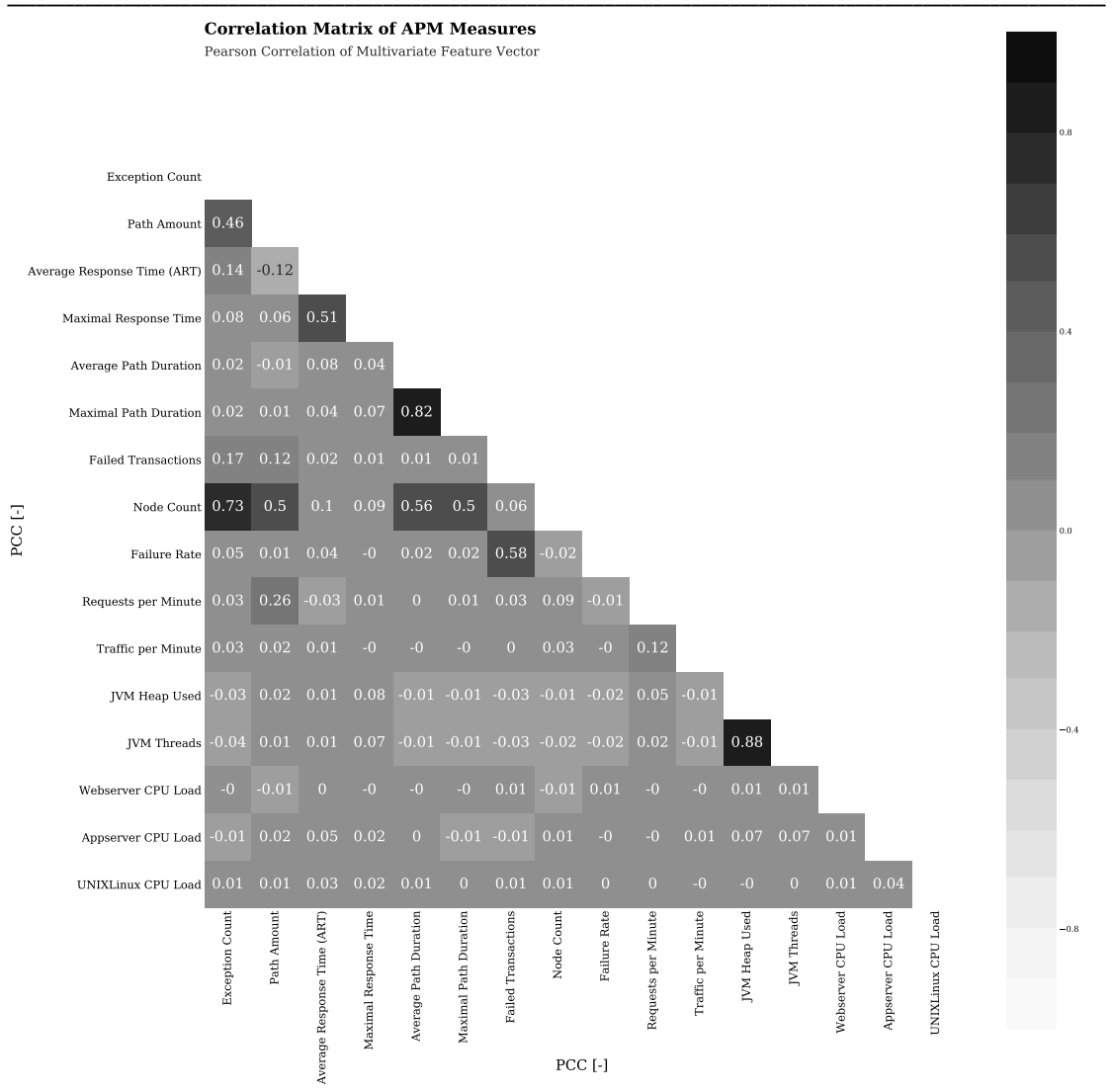


Figure 4.9.: Correlation matrix of APM measures for App_1

Feature Selection

In the previous paragraphs, the set of features is already narrowed down by analyzing autocorrelation and correlation among features. Nevertheless, from the identified four

feature sets (F_1 - F_4) of univariate and multivariate features, we aim to select the most relevant ones. Feature selection is performed by first training a model, in our case a random forest regressor, with the full set of features and then eliminating features based on their importances for the resulting model (Kohavi and John, 1997). In the implementation of the random forest estimator that we use, feature importances are defined as the sum over the number of splits (across all trees) that include the feature, proportionally to the number of samples it splits[3].

The final univariate feature set contains the last 120 minutes of ART (i.e., lag d=120) for respective applications. For the multivariate case, we identify five features for each of the applications through iteratively eliminating the least important features (see figure 4.6). The reason why we choose five features is pragmatic: as we lag the features (i.e., take multiple time steps as separate columns in our feature vector X), the dimensionality of X is the amount of features times the lag d which can result in high dimensional feature sets. In the present case, the best results are achieved with d=30 and a set of five features which eventuates into 150 columns. Hence, the final multivariate feature sets identified for both applications are listed in table 4.1. Except for Average ART Last Hour, all of these features are considered for the previous 30 minutes (i.e., d=30), to predict subsequent 30 minutes.

Table 4.1.: Final multivariate feature sets for enterprise applications

	App_1	App_2
Features	ART	ART
	Average ART Last Hour	Average ART Last Hour
	Maximal Response Time	Maximal Response Time
	Average Path Duration	Average Path Duration
	Nodes per Path	Application Server CPU Load

These feature sets are used and compared for the model implementations in the next subsection. After the feature engineering and selection, the data frames containing uni- and multivariate feature vectors X look as depicted in figure 4.10.

4.2.4. Model Implementations

This subsection describes the predictive models, which are implemented to forecast ART with 30 minutes time horizon. Req_3 constitutes each of the models to be comparable regarding different evaluation metrics. To take assorted aspects of predictive quality into consideration, five evaluation metrics are used to compare the models (Shcherbakov

[3]For the model building we use scikit-learn, a Python ML library, `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`, accessed on March 12, 2018

3 Data frame containing a feature vector of (self-)engineered and selected features

Univariate Feature Vector (lag = 120)

	ART(t-119)	...	ART(t-29)	...	ART(t)	ART(t+1)	...	ART(t+30)
90% Training Data	649.45	...	309.23	...	309.23	345.21	...	423.23
	291.23	...	451.34	...	451.34	425.26	...	542.76
	624.32	...	520.34	...	520.34	738.62	...	643.25
	...	...	...	...	...	...	...	...
	913.24	...	664.32	...	664.32	356.31	...	252.34
10% Test Data	1426.54	...	848.35	...	848.35	903.63	...	1034.25
	912.32	...	938.31	...	938.31	993.25	...	1144.28
			X (feature vector)			y (output vector)		

Multivariate Feature Vector (lag = 30)

	ART(t-29)	...	ART(t)	Nodes per Path (t-29)	...	Nodes per Path (t)	Average Path Duration (t-29)	...	...	ART(t+30)
90% Training Data	669.45	...	309.23	45.23	...	42.41	702.23	...	...	423.23
	301.23	...	451.34	40.21	...	46.75	412.21	...	...	542.76
	643.34	...	520.34	54.92	...	53.16	650.42	...	...	643.25
	...	...	...	...	...	...	...	...	...	...
	923.34	...	664.32	42.19	...	66.54	1023.55	...	...	252.34
10% Test Data	1223.52	...	848.35	44.20	...	41.23	1243.15	...	...	1034.25
	902.42	...	938.31	47.76	...	39.54	1002.42	...	...	1144.28
				X (feature vector)					y (output vector)	

Figure 4.10.: Data frame after feature engineering and selection

et al., 2013). In the formulas for the following metrics, $\hat{y}_t$ is the predicted value for observation t, y_t is the actual observed value, and n is the amount of observations:

- **Root Mean Squared Error (RMSE):** The RMSE is sensitive to outliers, as by squaring the residuals, larger errors have a disproportionately large impact on its score. Furthermore, it is scale-dependent and results subject to the fraction of data.

$$RMSE = \sqrt{\frac{\sum_{t=0}^{n}(\hat{y}_t - y_t)^2}{n}} \tag{4.3}$$

  The best score for RMSE is 0.

- **Root Mean Squared Percentage Error (RMSPE):** The RMSPE expresses accuracy in percentage which makes it scale-independent and thus, different data sets can be compared. However, drawbacks are that RMSPE cannot be used if there are zero values and is sensitive to outliers.

$$RMSPE = \sqrt{\frac{1}{n}\sum_{t=0}^{n}(\frac{y_t - \hat{y}_t}{y_t})^2} \tag{4.4}$$

The best score for RMSPE is 0, whereas the worst is 1.

- **Root Mean Squared Logarithmic Error (RMSLE):** The RMSLE is similar to RMSE, but it takes the logarithm of the predictions and actual values. It is commonly used if large differences in the predicted and the actual values should not be penalized when both, predicted and actual values are large numbers.

$$RMSLE = \sqrt{\frac{1}{n}\sum_{t=0}^{n}(\log\frac{\hat{y}_t+1}{y_t+1})^2} \tag{4.5}$$

  The best score for RMSLE is 0.

- **Mean Absolute Error (MAE):** The MAE is a scale-dependent metric, but large errors are not penalized as much as with RMSE.

$$MAE = \frac{1}{n}\sum_{t=0}^{n}|y_t - \hat{y}_t| \tag{4.6}$$

  The best score for MAE is 0.

- **Mean Absolute Percentage Error (MAPE):** The MAPE, as the RMSPE, expresses accuracy in percentage which makes it scale-independent. Major drawbacks are that MAPE cannot be used if there are zero values and that there is no upper threshold for forecasts which are too high.

$$MAPE = \frac{1}{n}\sum_{t=0}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \tag{4.7}$$

  The best score for MAPE is 0, whereas the worst is 1.

Since we are predicting 30 steps into the future, we get one evaluation metric score per predicted time step. With the aforementioned 10% test data, we create 6048 30-minute forecasts. Therefore, we receive one evaluation metric score, e.g., RMSE, for each of the 30 predicted time steps. For instance, $RMSE(1)$ is the RMSE of all the 6048 forecasts for the first minute ahead. As we outline the results of the different models in the following subsections, we always provide the graph of the RMSE for App_1 and draw a line for their mean and median respectively. The graphs for App_2 are provided in the appendix A.1. Since the scope of this subsection is limited, only RMSE is considered in detail as it is sensitive to outliers and the error has the same unit as the predictions. For the final evaluation in section 4.3, we take the mean of the evaluation metric scores for the 30 time steps for each of the five evaluation metrics.

In ML, it is common practice to use cross-validation techniques to obtain reliable estimates for the degree of model generalization (Kohavi, 1995; Raschka, 2017). We train the models on a previously defined training data set, which contains 90% of the original data, and use the remaining 10% for testing. However, to check generalization of the chosen model and to find the optimal set of hyperparameters for the models, we apply

k-fold cross-validation. The result is an estimated average performance E of the model, which equally considers the estimated performances of each of the five folds for each hyperparameter combination. We choose k=5 in alignment with Raschka (2017). After performing a so-called grid search for optimal model hyperparameters, we retrain the model on the whole training set and check its performance with the aforementioned evaluation metrics. Figure 4.11 illustrates the process of 5-fold cross-validation and grid search for hyperparameters. In the subsequent paragraphs, we always describe the best performing hyperparameter combination for the two previously defined uni- and multivariate feature sets.

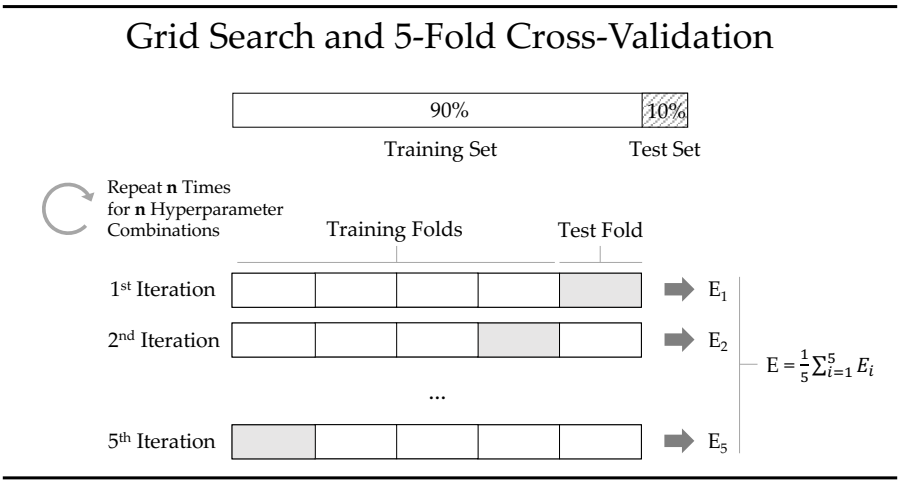


Figure 4.11.: Concept of hyperparameter grid search and 5-fold cross-validation

Baseline

To have an estimator, which serves as a starting point when comparing different models, it is reasonable to establish a baseline model at first. For our purpose, the best performing baseline model is to take 30 minutes of ART as the predicted values for subsequent 30 minutes of ART. This baseline model is more accurate than solely taking the previous minute and using its ART as a prediction for the entire subsequent 30 minutes. Figure 4.12 shows the RMSE of the baseline model.

Linear Regression

Linear regression is the only linear model that we apply to the forecasting problem. There are no hyperparameters we need to set and the best results are achieved with a direct multi-step forecasting strategy (i.e., training 30 independent models, see subsection 2.3.1). For the multivariate case, the features first need to be standardized to have a uniform

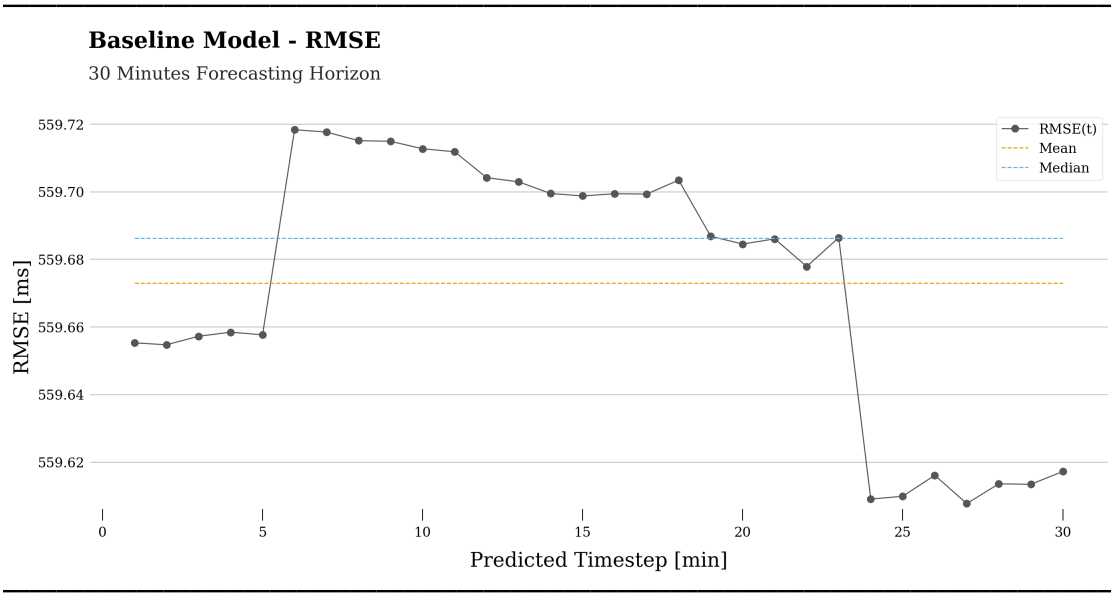


Figure 4.12.: RMSE baseline model for App_1

scale. Standardization for a feature X is done by subtracting the feature's mean value μ and dividing by the standard deviation σ:

$$Z = \frac{X - \mu}{\sigma} \tag{4.8}$$

Hence, Z is the standardized feature vector, which is fed into the model. Figure 4.13 and 4.14 show the RMSE of the linear regression model for the uni- and multivariate feature set.

Tree-Based Regression

The tree-based regression we perform is based on random forests, which are ensembles of decision trees (see subsection 2.3.2). The hyperparameters we set are the number of estimators (i.e., number of trees in random forest) and the maximal tree depth. The best accuracy is achieved by setting the number of estimators to 100 and leaving the maximal depth unset. This implies that nodes are expanded until all leaves are pure or all leaves contain less than two data samples. Regarding the multi-step ahead time series forecasting strategy, a MIMO strategy yields the best results. Figure 4.15 and 4.16 show the RMSE of the random forest regression model for the uni- and multivariate feature set.

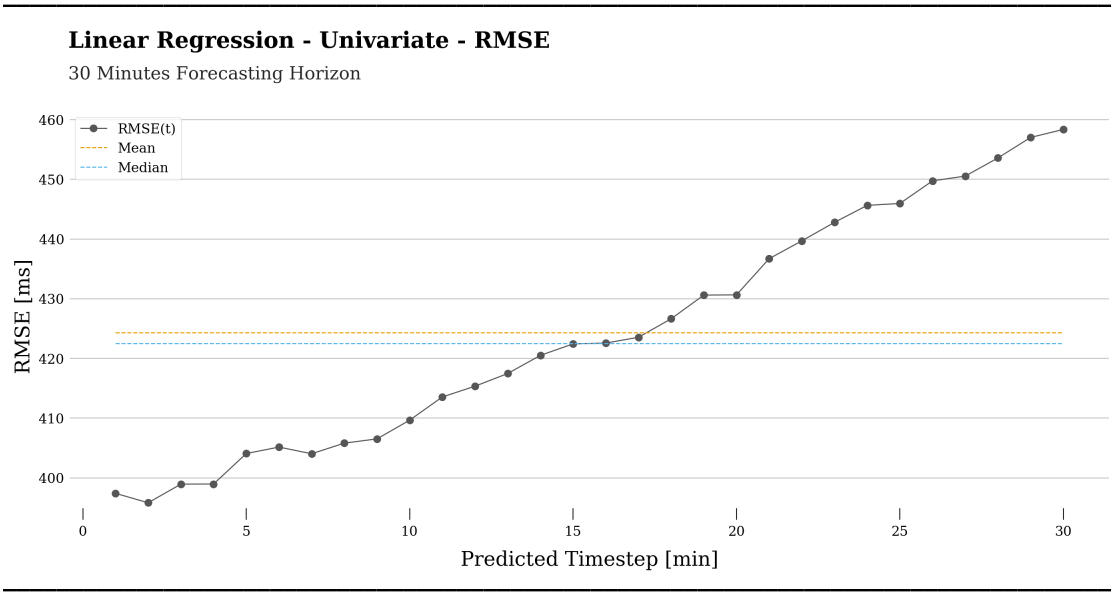


Figure 4.13.: RMSE linear regression model, univariate feature set for App_1

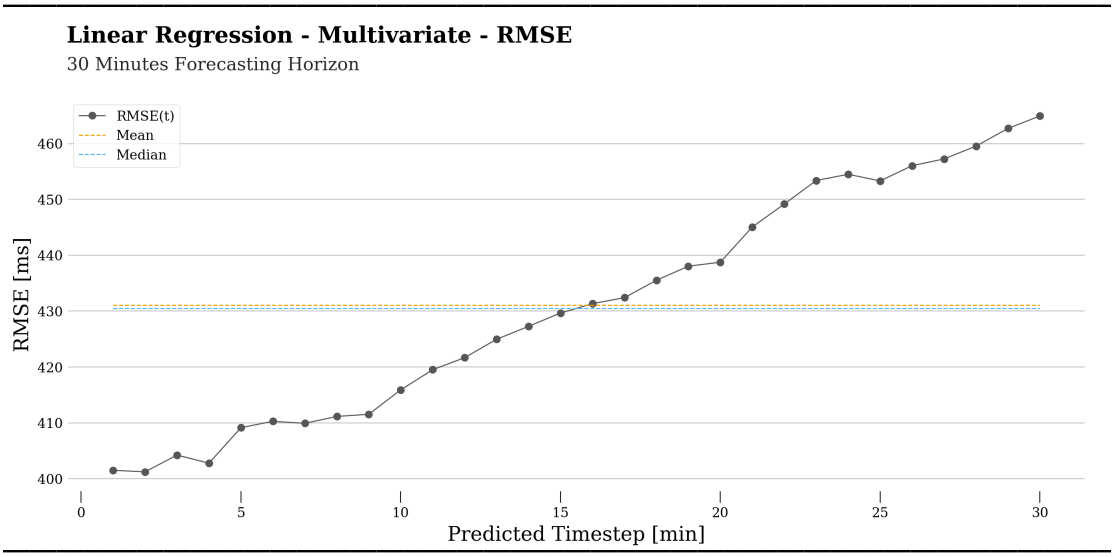


Figure 4.14.: RMSE linear regression model, multivariate feature set for App_1

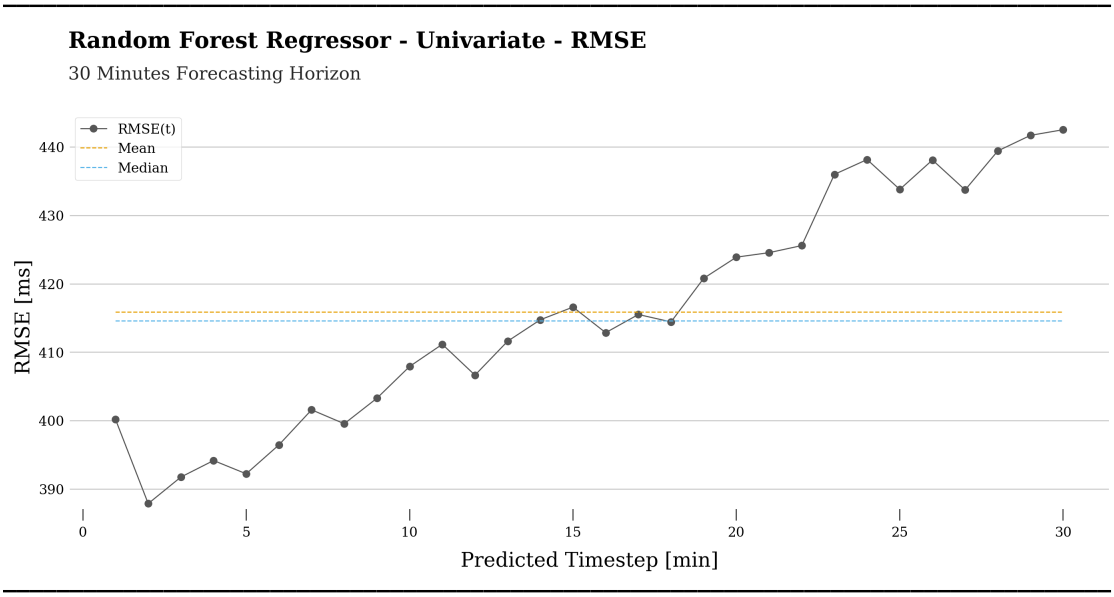


Figure 4.15.: RMSE random forest regression model, univariate feature set for App_1

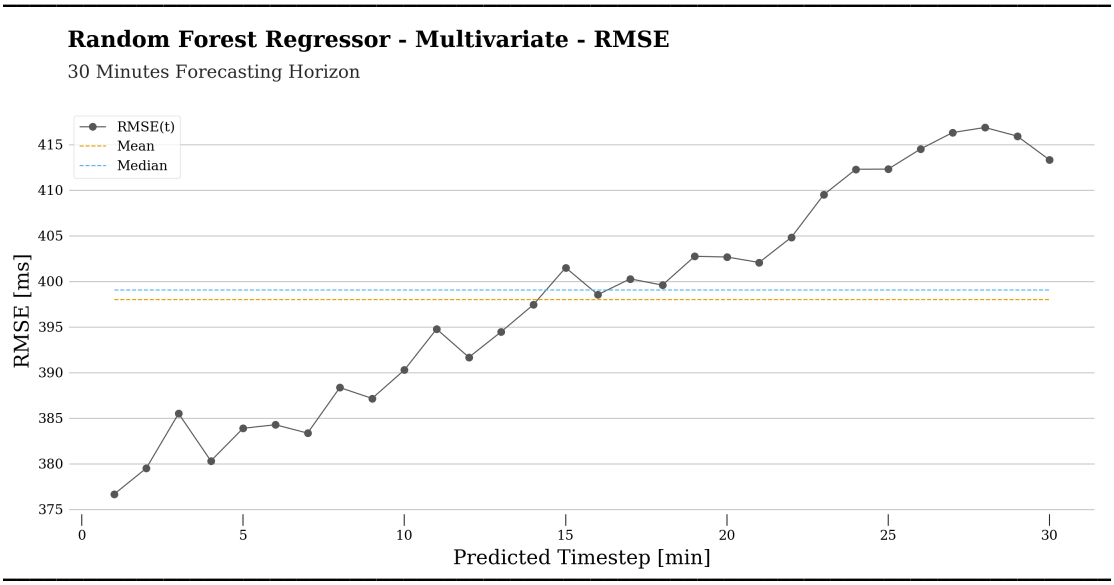


Figure 4.16.: RMSE random forest regression model, multivariate feature set for App_1

Multilayer Perceptron

MLPs can have multiple continuous outputs and thus are suitable for multi-step time series regression problems (see subsection 2.3.2). The network architecture we use for predicting the subsequent 30 minutes of ART by applying a MIMO strategy is illustrated in figure 4.17[4].

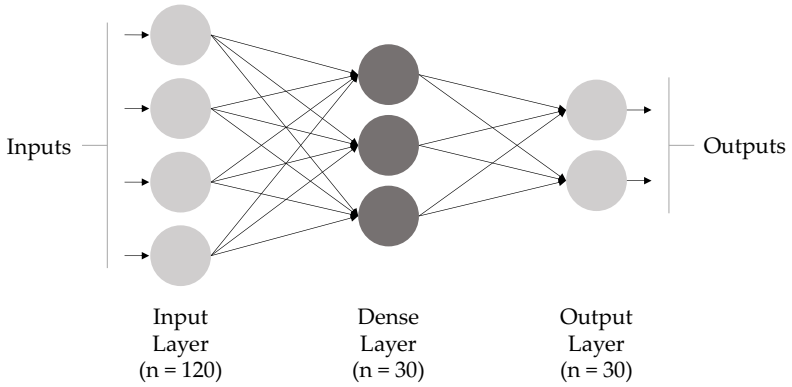


Figure 4.17.: MLP architecture: one hidden layer with 30 neurons and ReLU activation function

The hyperparameters we set are the batch size, the amount of epochs to train the model, the optimization algorithm, the loss function, and the amount of neurons and their activation function in the hidden layer. The set of best performing hyperparameters contains:

- **Batch Size:** 5
- **Epochs:** 10
- **Optimization Algorithm:** Adam[5]
- **Loss Function:** Mean Squared Error (MSE)
- **Amount Neurons in Hidden Layer:** 30
- **Activation Function for Hidden Layer:** ReLU

Additionally, in the multivariate case, the feature vector is standardized. Figure 4.18 and 4.19 show the RMSE of the MLP regression model for the uni- and multivariate feature set.

[4]For the model building for neural networks we use Keras, a Python ML library, `https://keras.io/`, accessed on March 12, 2018

[5]The Adam optimizer is a popular stochastic optimization algorithm for neural networks. See Kingma and Ba (2015) for details.

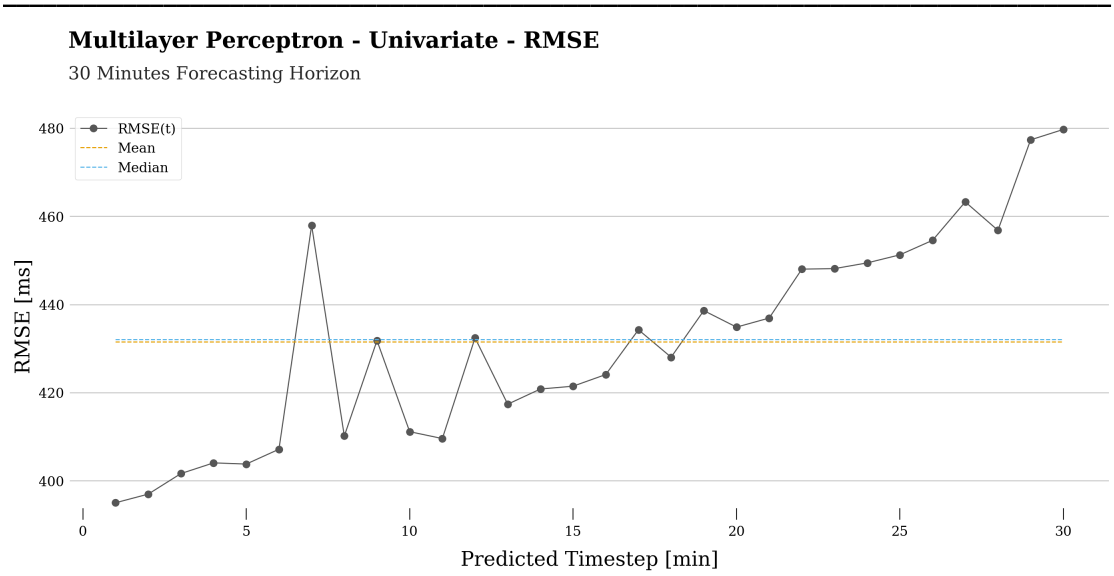


Figure 4.18.: RMSE MLP regression model, univariate feature set for App_1

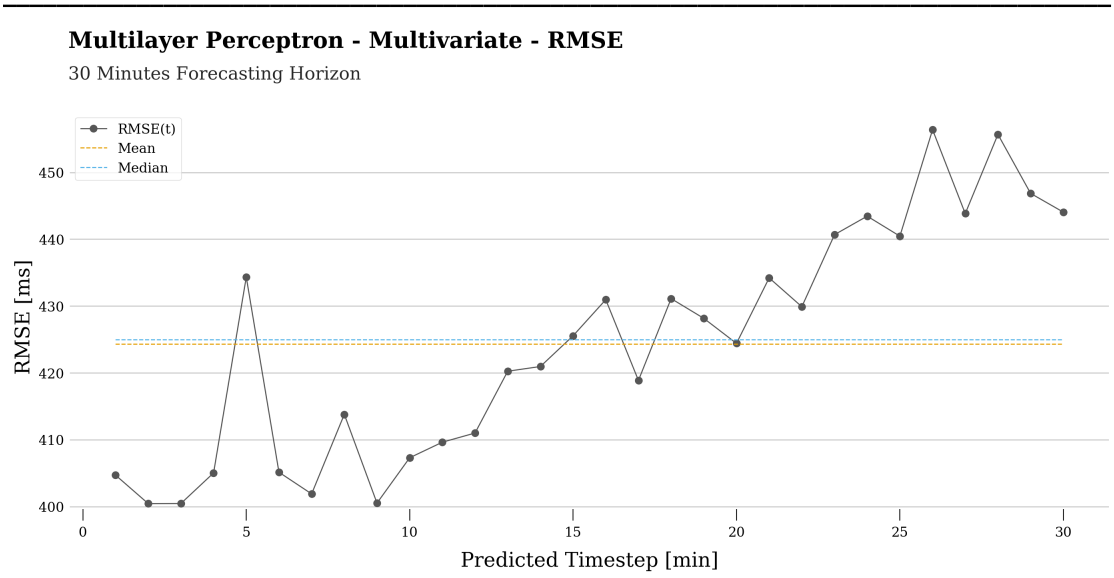


Figure 4.19.: RMSE MLP regression model, multivariate feature set for App_1

Long Short Term Memory networks

Besides the MLP, which is a feed-forward neural network architecture, RNNs with LSTM cells are implemented (see subsection 2.3.2). The network architecture we use for predicting the subsequent 30 minutes of ART by applying a MIMO strategy is illustrated in figure 4.20.

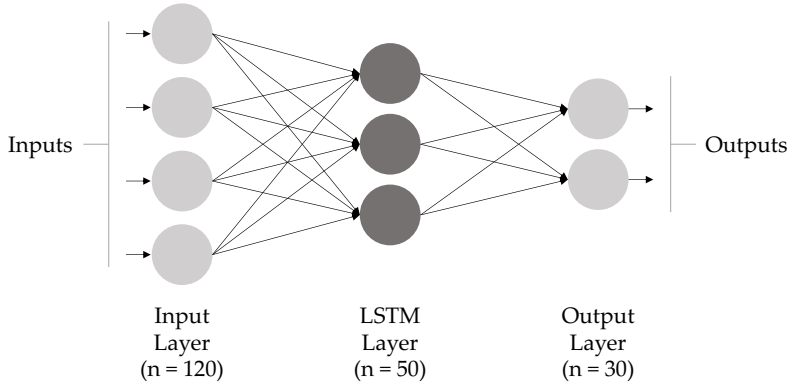


Figure 4.20.: LSTM architecture: one hidden layer with 50 LSTM units and ReLU activation function

The hyperparameters we set are the batch size, the amount of epochs to train the model, the optimization algorithm, the loss function, and the amount of neurons and their activation function in the hidden LSTM layer. The set of best performing hyperparameters contains:

- **Batch Size:** 30
- **Epochs:** 10
- **Optimization Algorithm:** Adam
- **Loss Function:** MSE
- **Amount Neurons in Hidden Layer:** 50
- **Activation Function for Hidden Layer:** ReLU

Additionally, in the multivariate case, the feature vector X is scaled to a value range between 0 and 1 by the applying the following transformation:

$$X_{scaled} = \frac{X - min(X)}{max(X) - min(X)} \tag{4.9}$$

Figure 4.21 and 4.22 show the RMSE of the LSTM regression model for the uni- and multivariate feature set.

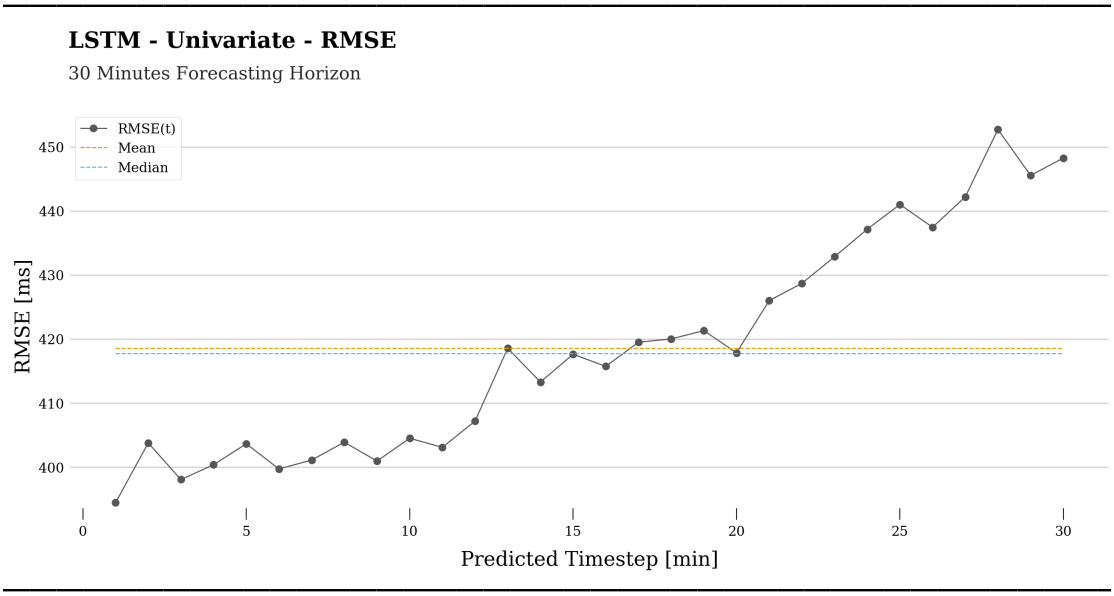


Figure 4.21.: RMSE LSTM regression model, univariate feature set for App_1

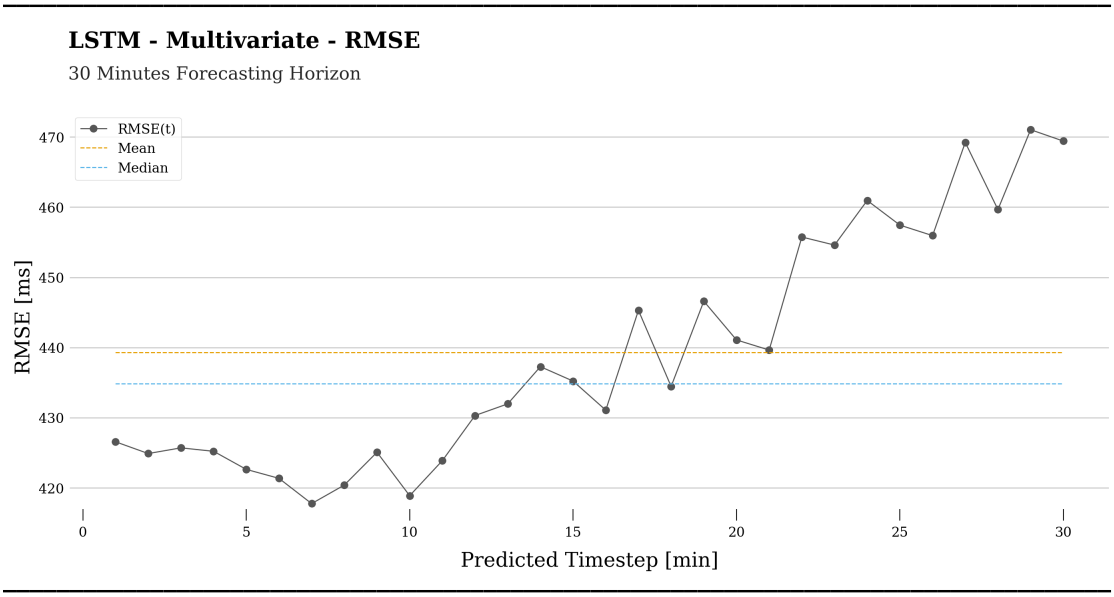


Figure 4.22.: RMSE LSTM regression model, multivariate feature set for App_1

4.3. Evaluation

In this section, we evaluate the previously described predictive models for forecasting the ART of App_1 and App_2. First, results of predictive models on the univariate feature set are summarized. Second, we delineate the results on the multivariate feature set. Third, we conclude the outlined evaluation and interpret the results with respect to $RQ_{2.2}$ and RC_1.

4.3.1. Results: Evaluation of Models on Univariate Feature Set

As aforementioned and in alignment with Req_3, we compare the models regarding their predictive accuracy with five different evaluation metrics (i.e., prediction error metrics) on the same test data set (i.e., 10% of the total data set). The total prediction error PE is obtained by averaging the errors PE_t of every of the 30 predicted time steps:

$$PE = \frac{\sum_{t=1}^{30} PE_t}{30} \tag{4.10}$$

Certainly, this is a debatable assumption which is further discussed in section 4.4.

Table 4.2 contains the evaluation results of the five different models, including the baseline model for App_1. Whereas table 4.3 depicts these results for App_2.

Table 4.2.: Comparison of prediction evaluation metrics of models for App_1, univariate feature set

	Baseline Model	Linear Regression	Random Forest Regressor	MLP	LSTM
MAE $[ms]$	266.29	212.43	205.64	220.60	213.81
MAPE $[\frac{\%}{100}]$	0.30	0.25	0.25	0.27	0.26
RMSE $[ms]$	559.67	424.29	415.88	431.55	418.54
RMSLE $[log(ms)]$	0.42	0.33	0.32	0.33	0.33
RMSPE $[\frac{\%}{100}]$	0.85	0.76	0.76	0.80	0.74

4.3.2. Results: Evaluation of Models on Multivariate Feature Set

Similar to the univariate case, the evaluation metrics for the multivariate models are summarized in table 4.4 for App_1 and in table 4.5 for App_2.

Table 4.3.: Comparison of prediction evaluation metrics of models for App_2, univariate feature set

	Baseline Model	Linear Regression	Random Forest Regressor	MLP	LSTM
MAE $[ms]$	195.67	173.45	164.49	160.24	156.22
MAPE $[\frac{\%}{100}]$	0.36	0.33	0.31	0.25	0.26
RMSE $[ms]$	366.95	277.87	269.07	300.33	292.15
RMSLE $[log(ms)]$	0.48	0.39	0.37	0.42	0.39
RMSPE $[\frac{\%}{100}]$	0.69	0.40	0.40	0.34	0.32

Table 4.4.: Comparison of prediction evaluation metrics of models for App_1, multivariate feature set

	Baseline Model	Linear Regression	Random Forest Regressor	MLP	LSTM
MAE $[ms]$	266.29	223.81	196.97	220.28	233.64
MAPE $[\frac{\%}{100}]$	0.30	0.27	0.23	0.27	0.29
RMSE $[ms]$	559.67	431.05	398.02	422.86	439.65
RMSLE $[log(ms)]$	0.42	0.34	0.30	0.33	0.35
RMSPE $[\frac{\%}{100}]$	0.85	0.76	0.70	0.78	0.76

Table 4.5.: Comparison of prediction evaluation metrics of models for App_2, multivariate feature set

	Baseline Model	Linear Regression	Random Forest Regressor	MLP	LSTM
MAE $[ms]$	195.67	267.20	163.17	177.89	148.42
MAPE $[\frac{\%}{100}]$	0.36	0.56	0.31	0.34	0.22
RMSE $[ms]$	366.95	420.72	269.86	282.98	301.33
RMSLE $[log(ms)]$	0.48	1.27	0.37	0.47	0.41
RMSPE $[\frac{\%}{100}]$	0.69	0.89	0.40	0.44	0.29

4.3.3. Interpretation of Results

To interpret the results with respect to what $RQ_{2.2}$ and RC_1 constitute, we emphasize the following aspects in this subsection:

1. **Comparison Linear to Nonlinear Models:** Outline the predictive accuracy of implemented linear models compared to nonlinear models.
2. **Univariate vs. Multivariate Features:** Compare the results of models using uni- and multivariate feature sets, respectively.
3. **Analysis of Conspicuous Evaluation Metrics:** Interpret conspicuous evaluation metrics by incorporating their inherent characteristics, e.g., outlier sensitivity.
4. **Definition of Research Artifact:** Choose the best performing implemented model for predicting response time of enterprise applications.

Comparison Linear to Nonlinear Models

In subsection 4.2.4, we start off with a baseline model, which serves as a benchmark for the subsequently developed linear and nonlinear models. This baseline model prediction is outperformed by every model we implement.

> Each implemented model outperforms the baseline model which takes the previous 30 minutes as a prediction for the subsequent 30 minutes.

The only linear model implemented is a linear regression model. In our case, the random forest regressor outperforms the linear regression for every evaluation metric in both, the uni- and multivariate case for App_1 and App_2. MLP and LSTM are not as superior as the random forest regressor, but are at least equally good if not better than the linear regression. However, this is certainly also due to the limitations of this implementation which are further discussed in section 4.4. By further tuning the neural networks, their performance is presumably improved whereas the linear regression does not specifically allow additional hyperparameter tuning.

> Nonlinear models show superior predictive accuracy in comparison to linear models for forecasting the ART in enterprise applications.

Univariate vs. Multivariate Features

As aforementioned, we use uni- and multivariate feature sets and compare the models built on top of them regarding their predictive accuracy. The results show that the best results for App_1 are obtained with the multivariate feature set. For App_2, only the

RMSLE has a marginally better result for the univariate feature set, all the other metrics yield the best model for multivariate features. However, when comparing the same model, e.g., LSTM for App_1, for uni- and multivariate features, we find that there is no definiteness of superior multivariate models.

> The best model for both applications is built on a multivariate feature set which implies that other enterprise application measures have causal dependencies to the ART.

Admittedly, this could be reasoned in deficient data preparation for multivariate features in neural networks. Selecting and correctly preparing multivariate features is challenging and demands rigorous investigation. We use the described standardization and scaling techniques for the feature vector, but more thorough investigation is beyond the scope of this thesis. Last, even though the response time forecasting problem is formulated as a generic regression problem, the results for the two applications are dissonant. Therefore, an important insight is that accurately forecasting the ART for enterprise applications is not application agnostic, but requires application-specific knowledge and investigation.

> Building accurate forecasting models for the ART of enterprise applications requires application-specific considerations and knowledge.

Analysis of Conspicuous Evaluation Metrics

Furthermore, it is conspicuous that the scores of RMSPE and RMSE are comparably high. In our case, the RMSE is approximately twice the MAE, which is – as the RMSE – also in the same unit as the predictions. Since RMSE penalizes outliers, we infer that the predictions are relatively inaccurate for extreme values. As RMSPE is scale-independent, we can compare it with a another real-world data set from Kaggle[6], where the RMSPE is between 0.1 and 0.2. Whereas our RMSPE scores are at approximately 0.7 for App_1 and 0.3 for App_2. Again, RMSPE is also sensitive to outliers which is an explanation for this divergence.

> Evaluation metrics which penalize large errors show that the predictions deviate significantly from extreme observations.

[6]Kaggle is a platform where ML competitions on real-world data sets are hosted, `https://www.kaggle.com/c/rossmann-store-sales/leaderboard`, accessed on March 14, 2018

Definition of Research Artifact

The models which yield the best evaluation metrics are dependent on the application either the random forest regressor or the LSTM model. Even though LSTMs are successfully applied in other time series forecasting problems (see subsection 2.3.2, we find that in our case the random forest regressor outperforms the LSTM model in 65% ($\frac{13}{20}$) of the cases (five evaluation metrics on uni- and multivariate feature sets for two applications). We put emphasis on the RMSE to specifically penalize outliers. The random forest regressor performs better than the LSTM model in all four cases (uni- and multivariate feature set for two applications times). It is noteworthy, that not much model tuning is done on any of the models. As a result, we constitute the first research artifact as a random forest regressor prediction model to forecast the response time of enterprise applications.

> The first research artifact is a random forest regressor using a multivariate set of features for multi-step forecasting of ART for enterprise applications.

Summary

Table 4.6 summarizes the results of the evaluation.

Table 4.6.: Summary of evaluation results

	Results
1. Comparison Linear to Nonlinear Models	• All implemented models outperform baseline model • Nonlinear models have superior predictive accuracy
2. Univariate vs. Multivariate Features	• Multivariate feature set yields better predictions • Application-specific knowledge indispensable for feature engineering
3. Analysis of Conspicuous Evaluation Metrics	• Predictions deviate significantly from extreme observations
4. Definition of Research Artifact	• Best model is random forest regressor on multivariate feature set

4.4. Discussion & Conclusion

In this section, we discuss the evaluation results of RC_1 and $RQ_{2.2}$ and the limitations of the applied approach which are beyond the scope of this thesis.

First, we find that the predictions from our models are deviating significantly from extreme observations. This is a major shortcoming as in practice, these observations or points in time are of particular interest to IT operations departments. Generally, training the models on larger amounts of data where outliers are more prevalent is a way to address this problem. Hence, the robustness and generalization are a limitation of the implemented models where more training data could potentially yield better results.

Second, the multi-step forecasts are evaluated by taking the average prediction error of the 30 predicted time steps (i.e., minutes). This is a straightforward approach, but it does not take into account the distribution of errors. Hence, we do not consider maximum, minimum, or median of the prediction errors for a particular step (i.e., minute). To address this limitation other aggregation functions could be considered, e.g., the median of the 30 prediction errors. However, this does not improve the multi-step forecast which – as expected – becomes worse the more steps we look into the future, but only provides a different interpretation of evaluation metrics.

Third, the feature engineering process yields no seasonalities or trends and only a rather insignificant daily cycle in the observed time series of the ART. Therefore, the data is characterized by noise and randomness and not by explainable time dependent components. This makes it hard to extract and generate meaningful features from the historical data which ultimately results in less accurate predictive models. As this limitation is inherent to the collected monitoring data for respective applications, it is difficult to tackle this problem. By further parameterizing the predictive models, these could reflect noise in the training data more precisely, but this comes at the cost of over-fitting the training data leading to worse prediction accuracy on the test data set.

Fourth, in ML models hyperparameter tuning is a naturally a task of arbitrary duration and intensiveness. The scope of this thesis is limited to a 5-fold cross-validation with parameter grid search. However, the investigated models can certainly be further optimized. Emphasis is put on the non-exhaustive possibilities regarding neural network architectures (e.g., depth, width, or activation functions). Future work could build on top of the rather modest MLP and LSTM architectures.

Fifth, the feature engineering and selection process is potentially never-ending. There are always more latent variables to detect in the data and especially in multivariate analyses the amount of possible features is tremendous. We have outlined the chosen approach to define uni- and multivariate feature sets, but there are techniques beyond correlation analysis to find latent variables for feature engineering, e.g., PCA or Non-Negative Matrix Factorization (NMF).

Sixth, one of the insights we gain from the evaluation of the two investigated enterprise applications is that predicting performance or more precisely response time is not an application agnostic task. As APM measures differ in their distributions and outlier density, it is difficult to solve the performance prediction problem in a generic, application agnostic way.

Seventh, response time might be the most prevalent performance indicator among the APM measures (see chapter 3), but its expressiveness regarding the status of an enterprise application is questionable. Characteristics as synchronous or asynchronous programming patterns used inside the code of the enterprise application have an impact on the response time. Therefore, in chapter 5, several APM measures are incorporated to detect anomalies on a higher level.

Eighth, the available real-world monitoring data defines the scope for the analyses in section 4.2. The developed models are thus evaluated and built on fifteen APM measures – as time series data – from two enterprise applications operated by a single company. Further research could evaluate the implemented models on more enterprise applications in different organizations, and harness other APM data sources for the model building. Another approach for predictive modeling might involve utilizing trace data, which holds less aggregated information at the price of intricacy and large volume of data.

Last, to approach the motivated change in paradigm towards proactive APM, we develop and evaluate predictive models for performance in enterprise applications. However, the potentially generated business value, e.g., greater revenues through cost savings or higher customer satisfaction, is yet to be evaluated. The conducted analyses can hence serve as a starting point for further investigation regarding the monetary business impact of proactive APM. Consequently, in future research, derived insights could exploit the aforementioned optimization potential in EAM, i.e., improving information flows between EA layers, and thereby increase the alignment of business and IT.

Abridged Answer to $RQ_{2.2}$

The analyses on forecasting the response time in enterprise applications based on historical APM measures yield that – in the present case – **random forest regressors** constitute the most accurate prediction models. The best predictions are obtained by harnessing the information of multiple APM measures through **multivariate modeling**. **Application-specific** considerations and knowledge are beneficial, if not indispensable, for feature engineering and selection in the process of model building and refinement. However, the models have **deficiencies** regarding the **prediction of extreme observations** and the accuracy decreases with **increasing forecasting horizon**.

Quantitative Results for Best Model: Random Forest Regressor (multivariate)

	MAE $[ms]$	MAPE $[\frac{\%}{100}]$	RMSE $[ms]$	RMSLE $[log(ms)]$	RMSPE $[\frac{\%}{100}]$
App_1	196.97	0.23	398.02	0.30	0.70
App_2	163.17	0.31	269.86	0.37	0.40

5. Anomaly Detection in Application Performance Management

In this chapter, we elaborate on RQ_3 and develop RC_2, an implementation of a multivariate density-based anomaly detection model, which indicates the root cause for abnormal system behavior. Hence, the objects of investigation are anomalies in enterprise applications. In section 2.4, we define them as patterns that do not reflect conformity compared to an observed normal behavior (Chandola et al., 2009). These anomalies or abnormal behaviors of IT systems are henceforth referred to as abnormal system behavior. Abnormal system behavior is a subjective term since the degree of abnormality of a system state is relative. As outlined in subsection 2.1.2, practitioners in APM and APM software prioritize different system aspects to determine whether abnormal system behavior is prevalent. Depending on which APM measures are considered, these comprise (i) performance issues (e.g., high response time), (ii) outages (e.g., service unavailable), or (iii) corrupt business transactions (e.g., business logic is erroneous). When building anomaly detection models for enterprise applications, it is important to be aware of the availability of labeled normal and anomalous data (Chandola et al., 2009). The industry partner, which provides the APM data, does not yet capture time periods of abnormal system behavior. As a result, we have the freedom to define abnormal system behavior in enterprise applications in consultation with respective APM experts for our purposes by combining (ii) and (iii):

> **Definition of Abnormal System Behavior:** State in an enterprise application when the system is either unavailable or core business logic is not working as expected.

Consequently, to evaluate the anomaly detection model to be developed, we need to acquire the missing labels of anomalous data. The solution we build is a web-based software, which enables those responsible for the development and operation of enterprise applications to manually label time periods of abnormal system behavior. In section 5.3, we describe in detail how data can be labeled in the provided solution.

Recapitulation RQ_3

How well can abnormal system behavior of enterprise applications be detected based on APM measures?

Recapitulation RC_2

Implementation of a multivariate density-based anomaly detection model providing an indication for the root cause of abnormal system behavior.

5.1. Define Objectives

As in chapter 4, we collect requirements for the developed solutions to address the scope of RC_2 and to answer RQ_3:

- **Req_1:** The model uses density-based unsupervised learning techniques to detect abnormal system behavior.
- **Req_2:** The model can take a multivariate feature vector of arbitrary dimensionality as input to incorporate multiple APM measures.
- **Req_3:** The model outputs a continuous measure, which reflects the degree of abnormality of the system over time. We call this measure the anomaly index and use it to evaluate the accuracy of the anomaly detection model.
- **Req_4:** The model provides a transparent indication for the root cause of detected abnormal system behaviors.
- **Req_5:** The software solution enclosing the anomaly detection model empowers practitioners to label time periods of abnormal system behavior to ultimately obtain a labeled data set, i.e., ground truth, of system anomalies.
- **Req_6:** The model can be compared and evaluated regarding its accuracy by comparing the outputted anomaly index to labeled time periods of abnormal system behavior.

Hence, the objective of this chapter is to elaborate on density-based anomaly detection techniques for enterprise applications with respect to these designated requirements. The research artifact for this contribution is defined as the most accurate implemented anomaly detection model.

5.2. Design & Development

This section encompasses the design and development process of the anomaly detection models. The monitoring data used for building the models is obtained from the APM systems of one highly business relevant enterprise application at a German car manufacturer. In the following, the application is depicted by App_1. Together

with experts from a specialized department for APM and those responsible for the development and operation of App_1, the scope of the contribution is defined as follows:

- **Time Period of Collected Monitoring Data:** Two months, October 1, 2017 to November 30, 2017
- **Input:** Fifteen APM measures with time series data of varying collection intervals, from 1 data point per minute to 1 data point per 60 minutes (step size defined as 1 minute)
- **Output:** A continuous measure reflecting the severity of abnormal system behavior, i.e., anomaly index, of the enterprise application over time.

The data is obtained from the same source monitoring systems as in chapter 4. Therefore, subsection 4.2.1 provides a thorough overview of the data set that is used.

As formulated in RC_2 and the requirements, we aim to develop models which are capable of detecting abnormal system behavior in an enterprise application through density-based unsupervised learning techniques. The approach we take is eminently inspired by three blog posts and one scientific paper:

- **Overseer Labs (2016 & 2017):** Overseer Labs[1] provide two blog posts where they describe how they use K-Means clustering, an unsupervised ML technique, to improve root cause analysis in case of system failures (Hasan, 2016; Hasan, 2017). They claim that individual measures can be very noisy and thus trigger a lot of false positives when it comes to alerting on univariate anomaly detection. Hence, they suggest an overall system health metric which incorporates several selected measures. The hypothesis is that this system health metric is a more accurate proxy with fewer false positives for abnormal system behavior. Ultimately, their goal is to facilitate root cause analysis by considering individual contribution of measures to a bad system health (i.e., low overall system health metric). The results are inconclusive as they struggle with noisy data and a lack of ground truth (i.e., labeled anomalous data) to evaluate and improve their model. Furthermore, they have to cope with model decay, where old data is less relevant than new observations and adjust the re-training of their model accordingly. Last, they are still seeking better incentives for users of their solution to label abnormal system states accurately.
- **Netflix (2015):** At Netflix[2], operations engineers have applied DBSCAN on server measures, i.e., univariate time series (Fisher-Odgen et al., 2015). Their objective is to detect unhealthy servers among the tens of thousands of servers Netflix' streaming service is running on. To incorporate domain knowledge, each service owner specifies the APM measure which is used to perform outlier detection through DBSCAN.

[1] `http://overseerlabs.io/`, accessed on March 16, 2018
[2] `https://www.netflix.com/`, accessed on March 16, 2018

- **Goldstein and Uchida (2016):** Goldstein and Uchida (2016) compare unsupervised ML outlier detection algorithms on multivariate data. The methodologies are tested on several data sets and yield that the kNN and LOF algorithm provide best results for global and local outlier detection, respectively. However, DBSCAN is not among the investigated algorithms.

We build our conceptual process for the design and development of RC_2 on the ideas from these sources. Thereby, it is reasonable to capture useful aspects of this former work into concrete implementation aspects. We structure these into Design Guidelines (DGs) with respect to the previously outlined requirements:

- **DG_1:** The major shortcoming of K-Means clustering is that the amount of final clusters has to be known in advance. DBSCAN determines the amount of clusters during run-time of the algorithm and yields good results for detecting abnormal system behavior (Fisher-Odgen et al., 2015). LOF, a density-based anomaly detection technique, performs well in related outlier detection problems (Goldstein and Uchida, 2016).

  Hence, we apply DBSCAN and LOF, two density-based unsupervised learning techniques, to address Req_1.

- **DG_2:** In accordance with Overseer Labs, we presume that an overall system health metric incorporating multiple application-specific APM measures is a better indicator for abnormal system behavior than single APM measures (Hasan, 2016).

  Thus, aligned with Req_2 and Req_3, we emphasize multivariate modeling and output an anomaly index which reflects the reverse of the overall system health.

- **DG_3:** As suggested by Netflix, to integrate application-specific knowledge for selecting APM measures for the clustering, we design a process which enables domain experts to manually select the relevant measures (Fisher-Odgen et al., 2015).

- **DG_4:** Based on Overseer Labs' approach towards root cause analysis, we use the individual contributions of the selected measures to the anomaly index to provide an indication about the root cause of an abnormal system behavior (Hasan, 2016).

- **DG_5:** To avoid a permanent lack of ground truth and to meet Req_5, we motivate practitioners to label time periods of abnormal system behavior through a simple, yet powerful labeling mechanism (Hasan, 2017).

The conceptual process is further illustrated in figure 5.1. Each phase in the outlined conceptual process can be interpreted as a separate software component. Implementation details on the different components are provided throughout the course of this section.

As a result, the solution we develop is a piece of software which encloses the research artifact, an anomaly detection model, and provides an interface for labeling abnormal

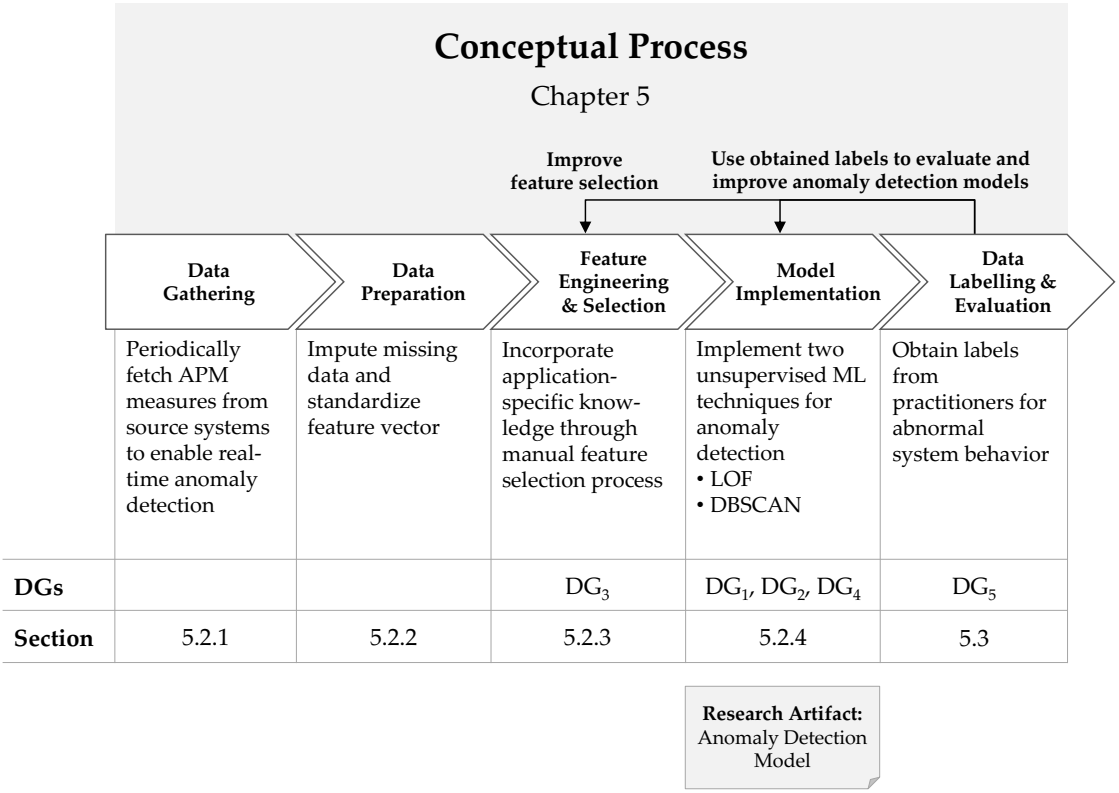


Figure 5.1.: Conceptual process of chapter 5 incorporating the defined DGs

system behavior. The software architecture and its components are defined by the requirements and DGs.

5.2.1. Data Source Identification

As aforementioned, the data base is the same as in chapter 4.

However, for testing the anomaly detection models in real-time, we need to automate and embed the process of data extraction, aggregation, and loading into the outlined software solution. Therefore, we introduce a software component which is responsible to periodically fetch newly available monitoring data and inform active clients, i.e., browsers which are currently using the software, about the data update. Typically, these periodic tasks are handled by task managers and schedulers. We use Apache Airflow[3], an open-source software package written in Python, which allows to programmatically author, schedule, and monitor workflows. The scheduler executes the task of retrieving new data every minute and then notifies currently active clients through web sockets.

[3] `https://airflow.incubator.apache.org/`, accessed on March 17, 2018

For the communication via web sockets, we use Socket.IO[4], an open-source software package written in JavaScript, which enables bi-directional communication between web clients and servers.

5.2.2. Data Preparation

As in chapter 4, the unprocessed data needs to be prepared for the modeling step. The data preparation encompasses the following two steps:

- **Data Imputation:** Data imputation is the task of filling missing data points with reasonable values. In the software solution we develop, the practitioner can select between mean and median filling. Similar to subsection 4.2.2, we receive the best results by filling missing values with the median value of the respective column (i.e., time series distribution).
- **Standardization:** The collected APM measures, i.e., time series, have different scales. Therefore, we need to standardize each of these potential features to have a similar scale. In chapter 4, we already applied the following technique to standardize a feature vector:

$$Z = \frac{X - \mu}{\sigma} \tag{5.1}$$

  where Z is the standardized feature vector which is fed into the model.

  Even though this is a common approach for standardization, we need to consider a major issue with this approach which Overseer Labs outline as well (Hasan, 2017). In case an APM measure has a very small standard deviation σ the division by this small value will result in large values for the standardized feature. Hence, the impact on the anomaly index is rather high which we should be aware of.

5.2.3. Feature Engineering

In subsection 4.2.3, we used backwards elimination of irrelevant features for filtering and selecting the ultimate set of (self-)engineered features. Since this requires building intermediate models where individual feature importance can be considered, we would need a ground truth, i.e., labels, to train a supervised classification model. However, we do not have labeled anomalous data initially, which limits the possibilities of feature engineering and selection. The approach we take is to apply unsupervised ML techniques. More precisely, we use the two density-based unsupervised learning algorithms, DBSCAN and LOF. Thus, there is no necessity for labeled abnormal system behavior as unsupervised ML intends to discover data patterns or interesting – but often latent – structures without a corresponding output variable, i.e., label. For our purpose, these

[4] `https://socket.io/`, accessed on March 17, 2018

latent structures potentially reveal abnormal system behavior in the form of outliers or anomalous data points.

Multivariate Features

The feature engineering step is similar as in subsection 4.2.3, where we enrich the set of fifteen APM measures by four self-engineered features:

- Average Failure Rate (i.e., ratio Failed Transactions to Path Amount)
- Average Exceptions per Path
- Average Failures per Path
- Average Nodes per Path

Feature Selection

Regarding the feature selection process, we rely on DG_3 which incorporates application-specific knowledge from practitioners. Therefore, the software solution we develop allows the so-called application owners, i.e., those responsible for the development and operations of the enterprise application, to manually select supposedly relevant features for the modeling step.

Figure 5.2 demonstrates the User Interface (UI) for the manual feature selection process inside the software solution. The available features are the standardized fifteen APM measures collected from the monitoring source systems and the four self-engineered features. The application owner can create configurations which are basically a self-selected set of features used for the model building. The example which is illustrated shows a configuration called TSSB_12 which contains the features Failure Rate, MQS Queuelength, Path Amount, and Average Response Time. Note that the terms monitors, measures, or metrics are often used interchangeably for APM measures in enterprise applications.

Consequently, we can make use of the obtained labels to improve the feature selection process. By performing a correlation analysis between the labeled time periods of abnormal system states and the twenty-two available features, we can infer the most relevant features for system abnormality. The best results for the anomaly detection models which are discussed hereinafter are achieved with the feature set listed in table 5.1. However, as section 5.3 shows, with only eight labeled abnormal system behaviors, the statistical relevance is not yet significant.

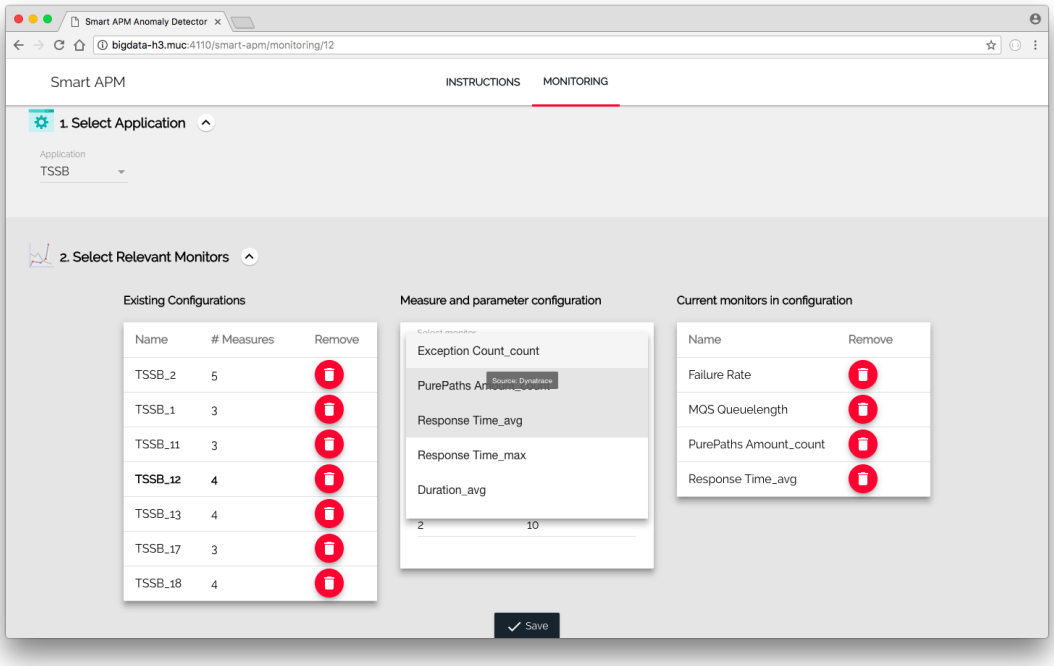


Figure 5.2.: UI for feature selection process

Table 5.1.: Best performing feature set for anomaly detection models

	App_1
Features	Average Path Duration
	ART
	Failure Rate
	Maximal Response Time
	Maximal Path Duration
	MQS Queuelength
	Failed Transactions
	JVM Threads

5.2.4. Model Implementations

This subsection describes the models which are implemented to detect abnormal system behavior in the aforementioned App_1. The developed model must meet the requirements Req_1, Req_2, Req_3, and Req_4. The result is a model incorporating DBSCAN which outputs what we call the anomaly index, a numerical measure reflecting the degree of system abnormality over time. However, in accordance with DG_1 we develop another anomaly detection model based on LOF, a technique which has been successfully applied in related outlier detection problems (Goldstein and Uchida, 2016). Hence, we have a generally accepted outlier detection algorithm which we can use as a reference model to evaluate the relevance of the generated anomaly index from the first model. Both of these models are described in detail subsequently.

Req_6 requires each of the models to be comparable regarding its accuracy. The ground truth which we compare against the outputted anomaly index from the models is obtained from the manual labeling process outlined in section 5.3.

DBSCAN

DBSCAN, as outlined in subsection 2.4.1, is a density-based clustering algorithm which reveals clusters in the data at points of high density. The approach we take consists of the following four steps:

1. Train DBSCAN model on multivariate (n-dimensional), standardized feature vector X with model parameters Eps and $minPts$. This step implements DG_1 and satisfies Req_1.
2. Select largest cluster (i.e., containing most observations) and set it as the cluster reflecting normal system behavior. Thus, define its centroid C_{normal} as the reference point for the calculation of the anomaly index. This step implements DG_2 and satisfies Req_2 and Req_3.
3. Calculate the euclidean distance $d = \sqrt{\sum_{i=1}^{n}(o_i - C_{normal_i})^2}$ from each observation o to the reference point C_{normal} in the n-dimensional space. The resulting numerical value is the anomaly index for the respective observation.
4. Register the relative contribution to the complete euclidean distance d of the n dimensions (i.e., features) for each of the observations. This step implements DG_4 and satisfies Req_4.

The procedure is exemplarily illustrated in figure 5.3 for the two-dimensional case. Note that each observation in the n-dimensional space is one point in time. Hence, the anomaly index can be considered over time by calculating the euclidean distance for each observation.

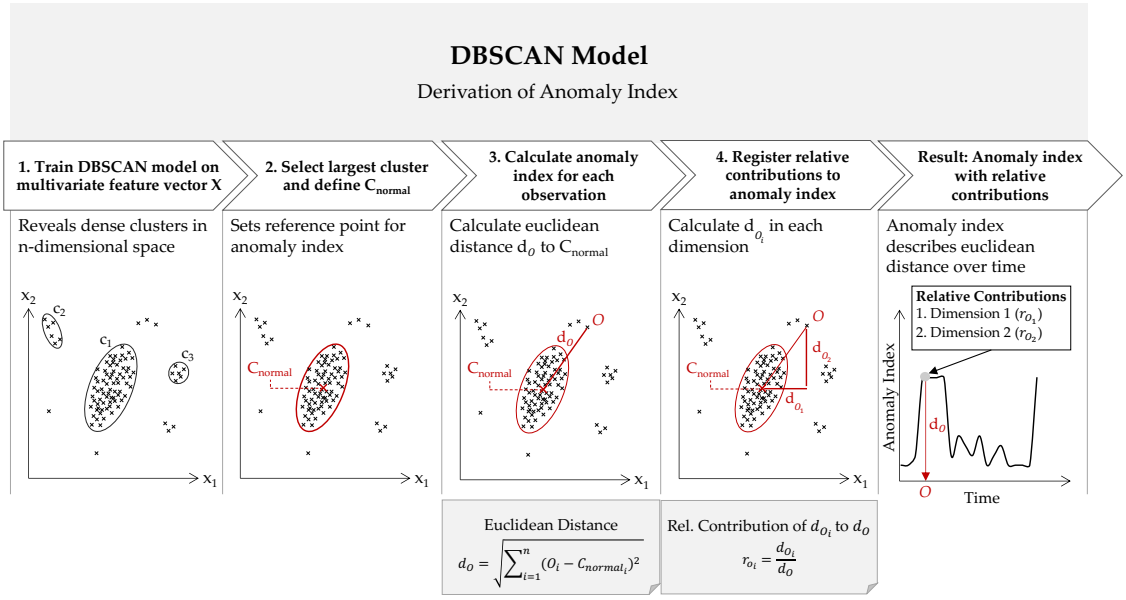


Figure 5.3.: Procedure of DBSCAN model: Derive anomaly index for each time step from euclidean distance to centroid of cluster reflecting normal system behavior

The hyperparameters we set are *Eps*, the radius of the density neighborhood, and *minPts*, the minimal amount of neighbors in the neighborhood. The set of best performing hyperparameters with respect to the evaluation metrics which are defined in the subsequent section 5.3 is as follows:

- **Eps:** 1
- **MinPts:** 10

LOF

LOF, as outlined in subsection 2.4.1, is a density-based outlier detection technique which assigns an outlier factor to each observation based on the density of its local neighborhood. The approach we take consists of the following 2 steps:

1. Train LOF model on multivariate (n-dimensional), standardized feature vector X with model parameter *minPts*. This step implements DG_1 and satisfies Req_1.
2. Calculate LOF of each observation o with equation 2.17. The resulting numerical value is the anomaly index for the respective observation. This step implements DG_2 and satisfies Req_2 and Req_3.

It becomes clear, that the LOF model cannot implement DG_4 and thus satisfy Req_4, as from the local density the individual contributions of the n dimensions to the anomaly index cannot be derived. Certainly, this is a major deficit compared to the DBSCAN

model. The procedure is exemplarily illustrated in figure 5.4 for the two-dimensional case. Note that each observation in the n-dimensional space is one point in time. Hence, the anomaly index can be considered over time by calculating the LOF for each observation.

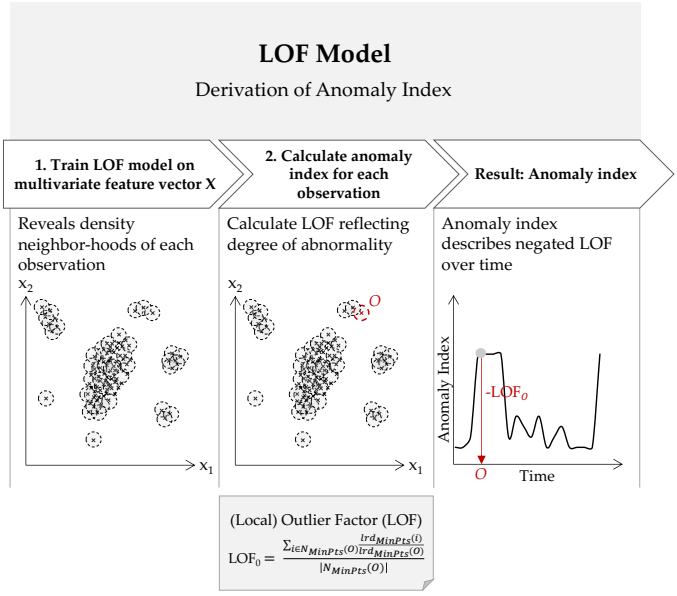


Figure 5.4.: Procedure of LOF model: Calculate outlier factor (i.e., anomaly index) from density of local neighborhood of each time step

The hyperparameter we set is *minPts*, the minimal amount of neighbors in the density neighborhood. The set of best performing hyperparameters with respect to the evaluation metrics which are defined in the subsequent section 5.3 is as follows:

- **MinPts:** 200

Software Components

The technical implementation of the model building is done by using the ML library scikit-learn as in chapter 4. Furthermore, the software architecture of the provided software solution which encloses the models is sketched in figure 5.5.

Accordingly, the software consists of the following components:

- **Frontend:** The frontend is a web UI built with Angular[5], an open-source framework for mobile and desktop frontend development written in Typescript. Additionally, for interactive visualizations, D3.js[6], a JavaScript charting library is used.

[5] `https://angular.io/`, accessed on March 24, 2018
[6] `https://d3js.org/`, accessed on March 24, 2018

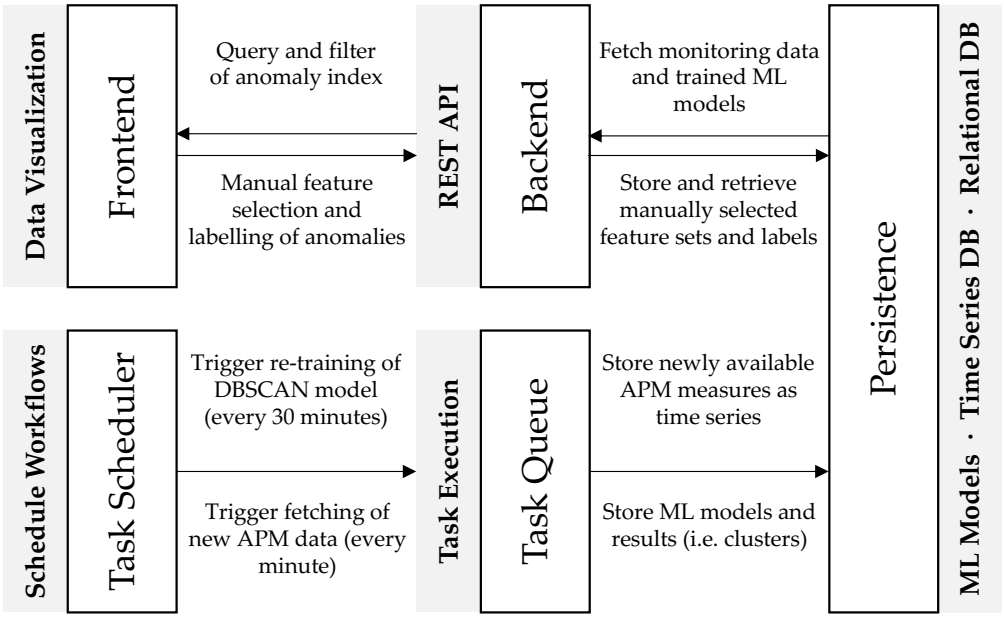


Figure 5.5.: Software components of developed software solution

The frontend enables the user to query and inspect the historical anomaly index (until current time), as well as labeling abnormal system behavior and manually selecting the features used for the model training as outlined in subsection 5.2.3.

- **Backend:** The backend delivers the content for the frontend through a REpresentational State Transfer (REST) API. Flask[7], an open-source package written in Python, serves as the framework for building this API. The REST API has endpoints for retrieving the anomaly index and managing the manually selected feature sets and labeled anomalies.

- **Task Scheduler:** As outlined in subsection 5.2.1, we use Apache Airflow for scheduling workflows. The re-training of the DBSCAN model is triggered every 30 minutes, whereas every minute we trigger the task of fetching newly available APM measures.

- **Task Queue:** For the tasks of model re-training and fetching new APM data, Celery[8], another open-source software package written in Python, is used. Celery is a an asynchronous task queue where tasks are executed concurrently on a single or more worker servers using multiprocessing.

- **Persistence:** To store the manually selected feature sets, the labeled anomalies, and the cluster centroid of the most recently trained DBSCAN model, we use a

[7] `http://flask.pocoo.org/`, accessed on March 24, 2018
[8] `http://www.celeryproject.org/`, accessed on March 24, 2018

PostgreSQL[9] database, which is an open-source object-relational database system. As aforementioned, the obtained time series APM data are stored into an InfluxDB. The ML models are serialized and stored on disk by using the Python module pickle[10].

Each of these components is implemented as a separate micro-service and Docker[11], a software which performs operating-system-level virtualization also known as containerization, is used to encapsulate these services into so-called containers. To enable communication among services and to orchestrate shared resources and virtual or physical volumes, Docker Compose[12], a tool for defining and running multi-container applications is utilized.

5.3. Evaluation

In this section, we evaluate the two developed models by considering various evaluation metrics and three different evaluation approaches. Furthermore, we elaborate on the process of obtaining the data labels which serve as the ground truth to estimate model accuracy.

5.3.1. Evaluation Metrics

In order to compare accuracy of anomaly detection models with respect to Req_6, there are several ways with different metrics which are applied in combination or separately. For classification tasks, the most commonly used metrics are precision, recall, F1-score, and Receiver Operating Characteristic (ROC) curve and the Area Under The Curve (AUC) (Powers, 2011). All of the metrics are based on the terms True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) which assess the results of classifiers by checking the predicted class against the true class. Table 5.2 clarifies the terminology.

Table 5.2.: Overview of binary classification terms

		Predicted Class	
		Positive	Negative
True Class	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

The evaluation metrics which are considered comprise the following Powers (2011):

[9] `https://www.postgresql.org/`, accessed on March 28, 2018
[10] `https://docs.python.org/3/library/pickle.html`, accessed on March 28, 2018
[11] `https://www.docker.com/`, accessed on March 24, 2018
[12] `https://docs.docker.com/compose/`, accessed on March 24, 2018

- **Recall:** The recall or True Positive Rate (TPR), is the ratio of TPs to the complete true class labels which is the sum of TPs and FNs:

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

  A perfect recall score is 1 whereas the worst score is 0.

- **Precision:** The precision or positive predictive value, is the ratio of TPs to all predicted true class labels which is the sum of TP and FP:

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

  A perfect precision score is 1 whereas the worst score is 0.

- **F1-Score:** The F1-score or F-measure, is the harmonic mean of precision and recall:

$$F1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.4}$$

  A perfect score for the F1-score is 1 whereas the worst score is 0.

- **False Positive Rate (FPR):** The FPR can be interpreted as the probability of false alarm and is the ratio of FPs to the sum of FPs and TNs:

$$FPR = \frac{FP}{FP + TN} \tag{5.5}$$

  A perfect FPR score is 0 whereas the worst score is 1.

- **AUC:** The ROC curve is created by plotting the TPR on the y-axis against the FPR on the x-axis at various threshold settings, where the threshold is incrementally increased from the lowest value of the observations to the highest. The AUC is the integral and thus depicts the surface area of the ROC curve:

$$AUC = \int_{\infty}^{-\infty} TPR(T)(-FPR'(T)), dT \tag{5.6}$$

  Figure 5.11 in subsection 5.3.3 delineates an example of a ROC curve. A perfect AUC score is 1 whereas the worst score is 0.

Goldstein and Uchida (2016) outline that AUC based evaluation has evolved to the standard for unsupervised anomaly detection. The reasons for that are that neither recall nor precision take into account the amount of TNs, and anomaly detection classification problems are typically imbalanced (Powers, 2011; Goldstein and Uchida, 2016). Therefore, we use AUC as our primary evaluation metric, but subsection 5.3.3 describes the threefold evaluation process in detail.

5.3.2. Data Labeling

As Req_5 and DG_5 postulate, the process of data labeling requires to be encapsulated in the developed software solution. Each obtained data label has a start time stamp, an end time stamp, and an optional description. Figure 5.6 illustrates the UI for manually adding these data labels. There, a so-called tag manager enables the application owner to both, creating new labels for abnormal system behavior and managing existing ones.

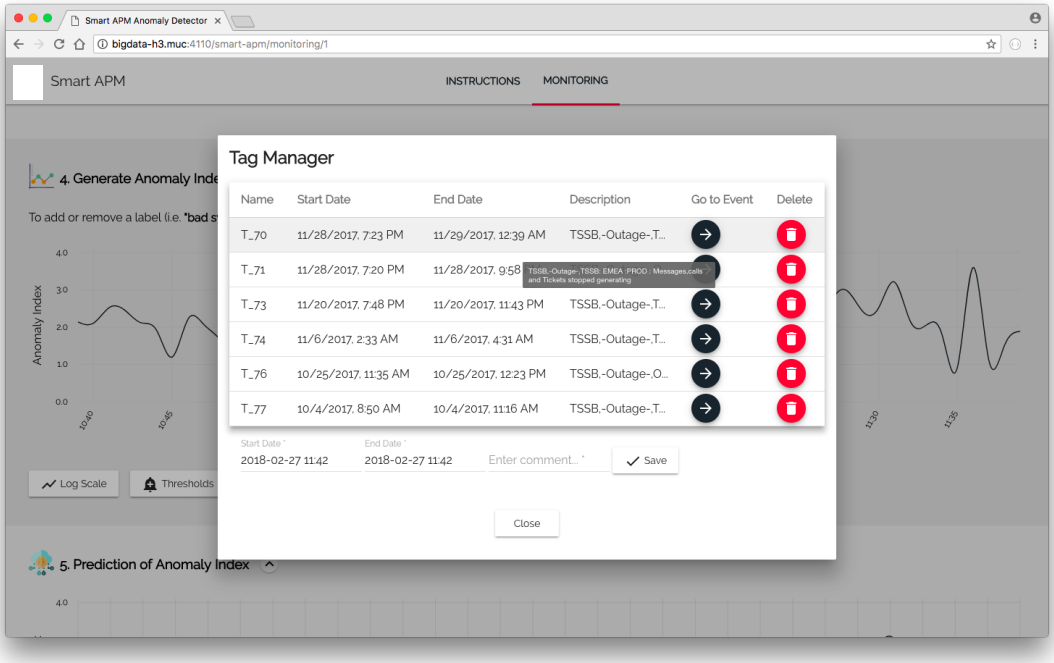


Figure 5.6.: UI for data labeling process

Then, these labels can be inspected by setting the time filter of the anomaly index graph to the respective abnormal time frame. This is shown in figure 5.7, where the points, i.e., minutes on the graph, which lie inside a labeled abnormal time period are drawn in red.

In total, eight data labels were obtained over the specified time period of two months. The following aggregation measures describe their distribution:

- **Min Duration:** 21 minutes
- **Mean Duration:** ~142 minutes
- **Max Duration:** 316 minutes

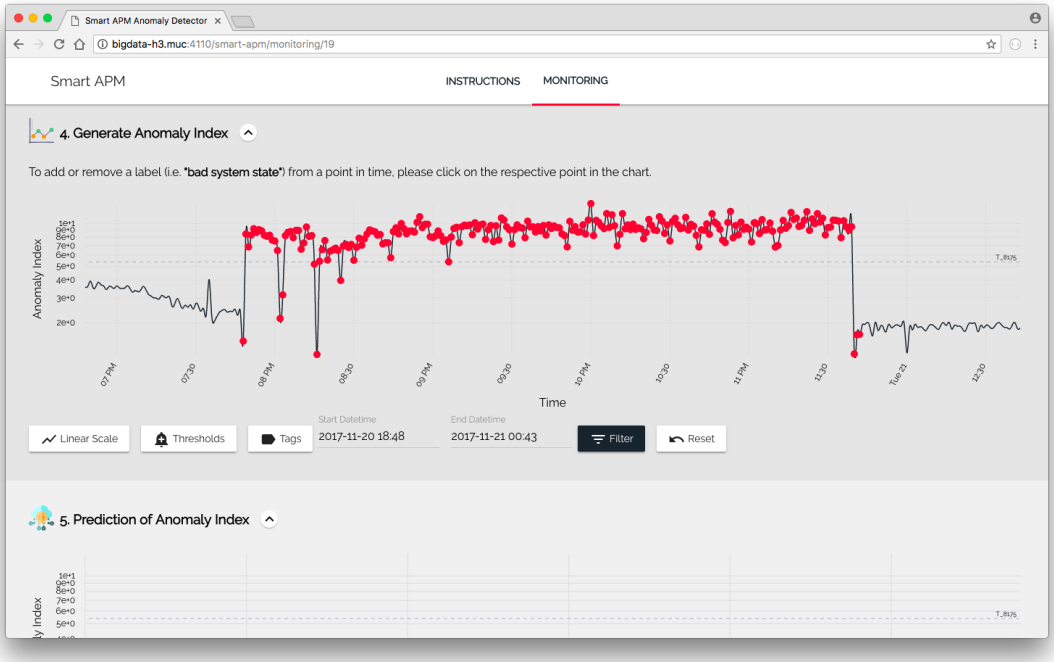


Figure 5.7.: UI for inspecting a labeled anomaly

5.3.3. Evaluation Approach

There are three types of detection for the obtained anomalous labels, an anomaly can either be completely detected, partly detected, or not detected with respect to the anomaly index and a defined critical threshold (see figure 5.8).

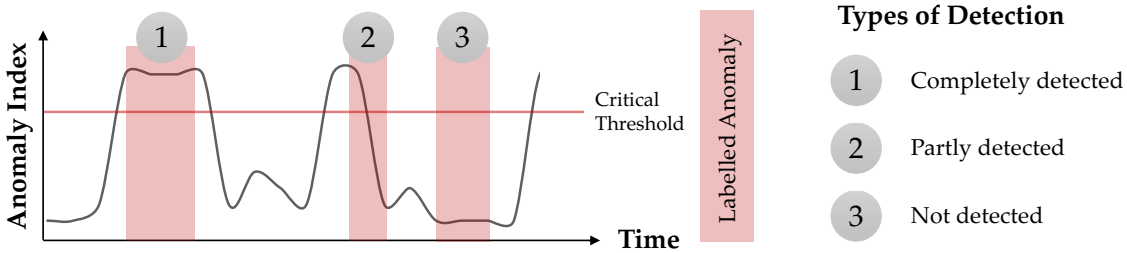


Figure 5.8.: Types of detection for labeled anomalies

The critical threshold can be defined manually in the UI of the software solution (see figure 5.9). However, we provide a recommendation for an alarm threshold $T_{\mu+3*\sigma} = \mu + 3 * \sigma$ which is derived from statistical process control and the six sigma principle (Montgomery, 2009).

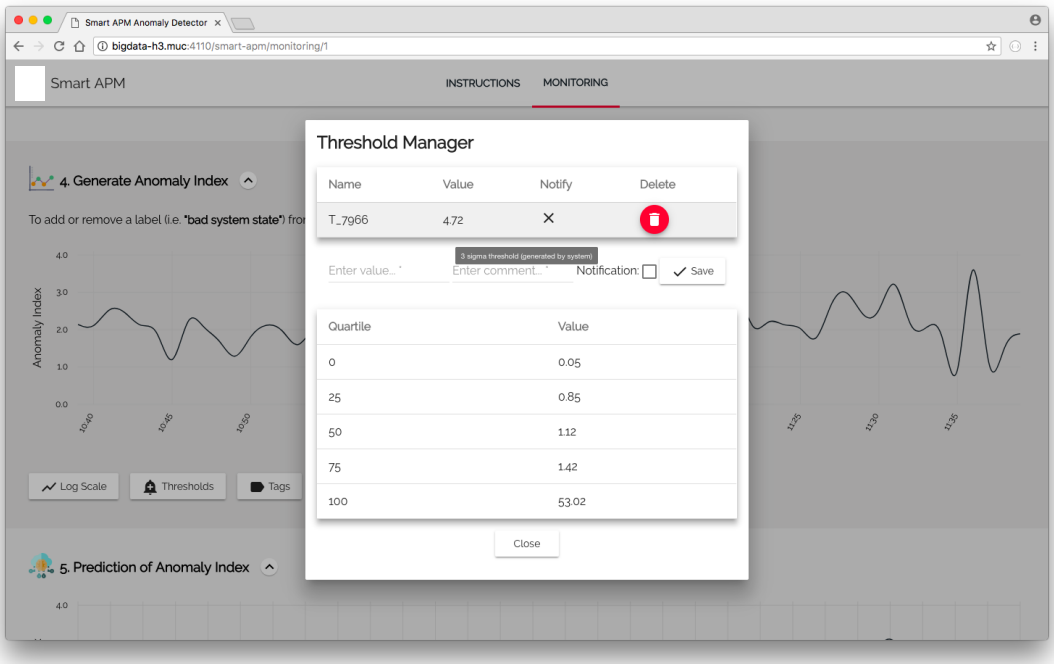


Figure 5.9.: UI for threshold management

In the following, we come up with three different approaches for evaluating the model accuracy and consecutively apply them to the two implemented models. We further use the critical thresholds $T_{\mu+3*\sigma}$ and $T_{99\%}$, where the second threshold is the value of the 99% percentile of the distribution of the anomaly index (Powers, 2011).

Label-Based Detection

Idea: The label-based detection approach treats all detections of type 2 as type 1 detections or more precisely only the maximal anomaly index within the labeled anomalous time frame is considered. Thus, the TPs are the sum of type 1 and 2 detections. Each undetected anomaly (i.e., type 3) counts as a FN whereas each value point outside a labeled time frame is either treated as TN or FP, in case the anomaly index exceeded the critical threshold. Hence, this approach is called label-based detection as it heavily depends on the obtained labeled time periods with their variable duration.

Problem: The problem with this approach is that on the one hand we count one TP for each correctly detected anomaly, whereas on the other hand, we count one FP for each value point exceeding the critical threshold. Hence, the traditional classification metrics are not suitable as the ratio of FP and TP is naturally imbalanced. Furthermore, we

introduce a bias by defining type 2 detections as type 1 only by looking at the maximal anomaly index within the anomalous time frame.

Result: The results of the label-based detection evaluation approach reveal the evaluation metrics shown in table 5.3.

Table 5.3.: Results of label-based detection evaluation for models

	DBSCAN Model	LOF Model
$AUC_{T_{\mu+3*\sigma}}$	0.97	0.98
$AUC_{T_{99\%}}$	0.97	0.98
$Recall_{T_{\mu+3*\sigma}}$	0.63	0.63
$Recall_{T_{99\%}}$	0.63	0.75
$Precision_{T_{\mu+3*\sigma}}$	0.01	0.01
$Precision_{T_{99\%}}$	0.01	0.01
$F1\text{-}score_{T_{\mu+3*\sigma}}$	0.01	0.01
$F1\text{-}score_{T_{99\%}}$	0.01	0.01

Figure 5.10 further illustrates the ROC curves of the two models.

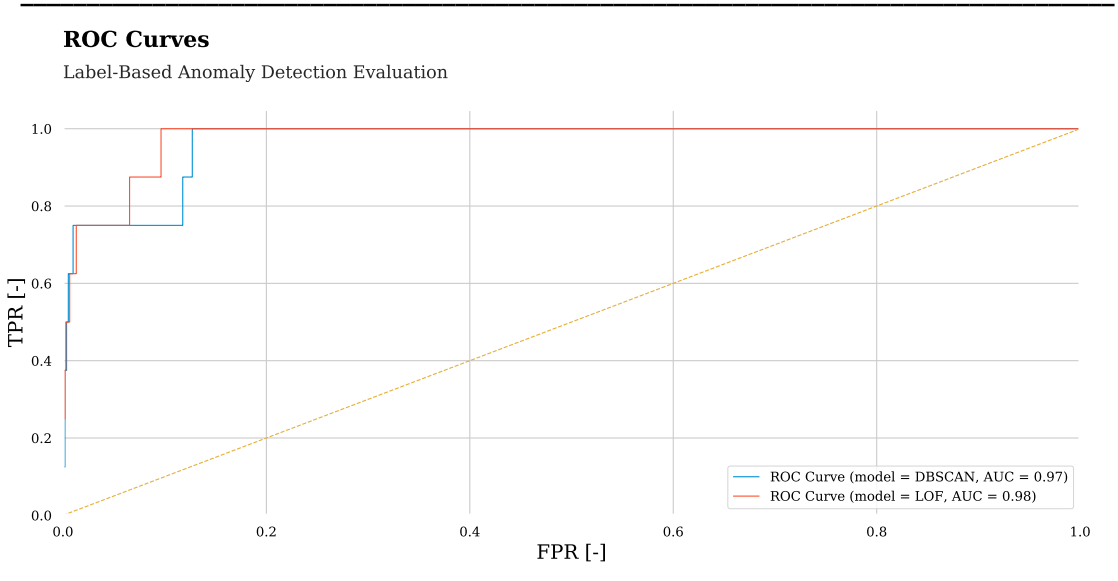


Figure 5.10.: Results of label-based anomaly detection evaluation for models

Point Anomaly Detection

Idea: The second approach evaluates each observation, i.e., anomaly index per minute, separately. Hence, we eliminate the shortcoming of the first approach regarding balance of classes. As aforementioned the most relevant metric we consider is AUC (Goldstein and Uchida, 2016).

Problem: The problem here is that APM measures tend to be volatile as we observed in chapter 4. As a result, the anomaly index presumably exceeds the critical threshold rather too often without an abnormal system state being prevalent. Thus, the implicit multiplicity of observations in a labeled anomalous time frame demands for evaluation of collective anomaly detection instead of detection of point anomalies.

Result: The results of the evaluation with respect to point anomalies reveal the evaluation metrics shown in table 5.4.

Table 5.4.: Results of point anomaly detection evaluation for models

	DBSCAN Model	LOF Model
$AUC_{T_{\mu+3*\sigma}}$	0.73	0.70
$AUC_{T_{99\%}}$	0.73	0.70
$Recall_{T_{\mu+3*\sigma}}$	0.04	0.02
$Recall_{T_{99\%}}$	0.07	0.06
$Precision_{T_{\mu+3*\sigma}}$	0.11	0.06
$Precision_{T_{99\%}}$	0.08	0.06
$F1\text{-}score_{T_{\mu+3*\sigma}}$	0.06	0.03
$F1\text{-}score_{T_{99\%}}$	0.07	0.06

Figure 5.11 further illustrates the ROC curves of the two models.

Collective Anomaly Detection (Sliding Window)

Idea: In contrast to the second approach, the third approach aims to consider collective anomalies rather point anomalies. However, different to the first approach, we set fixed time windows to evaluate the quality of detection of the models as proposed by Ahmad et al. (2017). These time windows of different window size are then slid across the entire observed time period of two months. As a result, we remove the problem of imbalance of the label-based approach and still consider collective anomalies.

Problem: The problem of induced bias from the label-based approach is still prevalent as we need to pick a strategy to treat type 2 detections either as type 1 or 3.

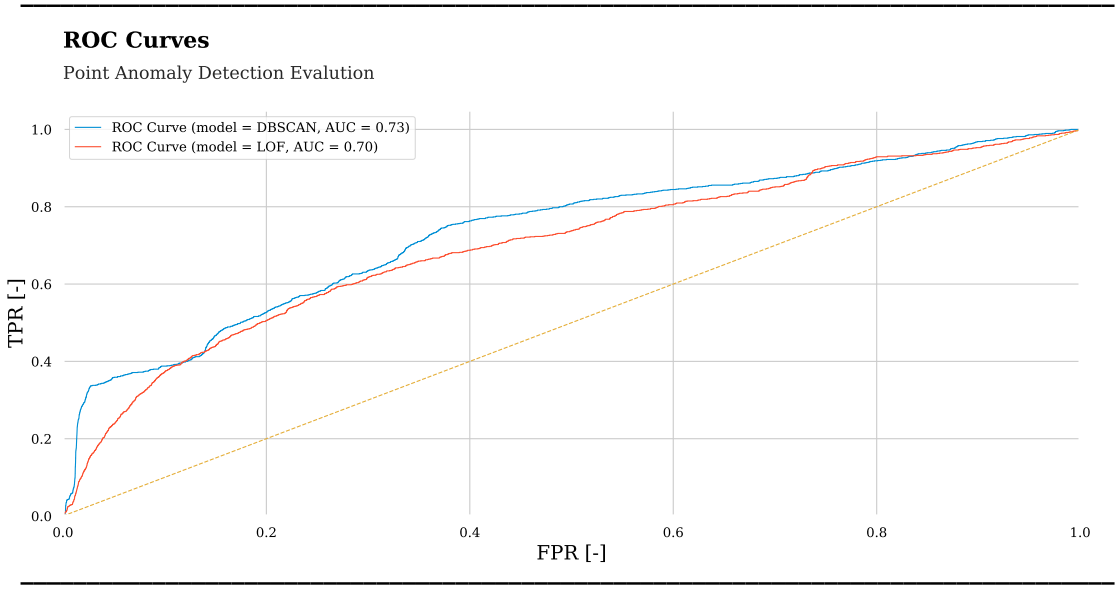


Figure 5.11.: Results of point anomaly detection evaluation for models

Result: The results of the evaluation with respect to collective anomalies across 50 evenly spaced window sizes from 1 minute to 300 minutes (rounded longest labeled anomalous time frame) are illustrated in figure 5.12 and 5.13.

5.3.4. Interpretation of Results

To interpret the results with respect to what RQ_3 and RC_2 constitute, we emphasize the following aspects in this subsection:

1. **Anomaly Detection Accuracy:** Outline how well the two implemented models detect abnormal system behavior with respect to the three evaluation approaches.
2. **Root Cause Analysis:** Compare the two implemented models regarding their usefulness for root cause analysis.
3. **Definition of Research Artifact:** Choose the best performing implemented model for detecting abnormal system behavior.

Anomaly Detection Accuracy

RQ_3 demands an evaluation of developed anomaly detection models to quantify how well they detect abnormal system behavior based on APM measures. In subsection 5.3.3,

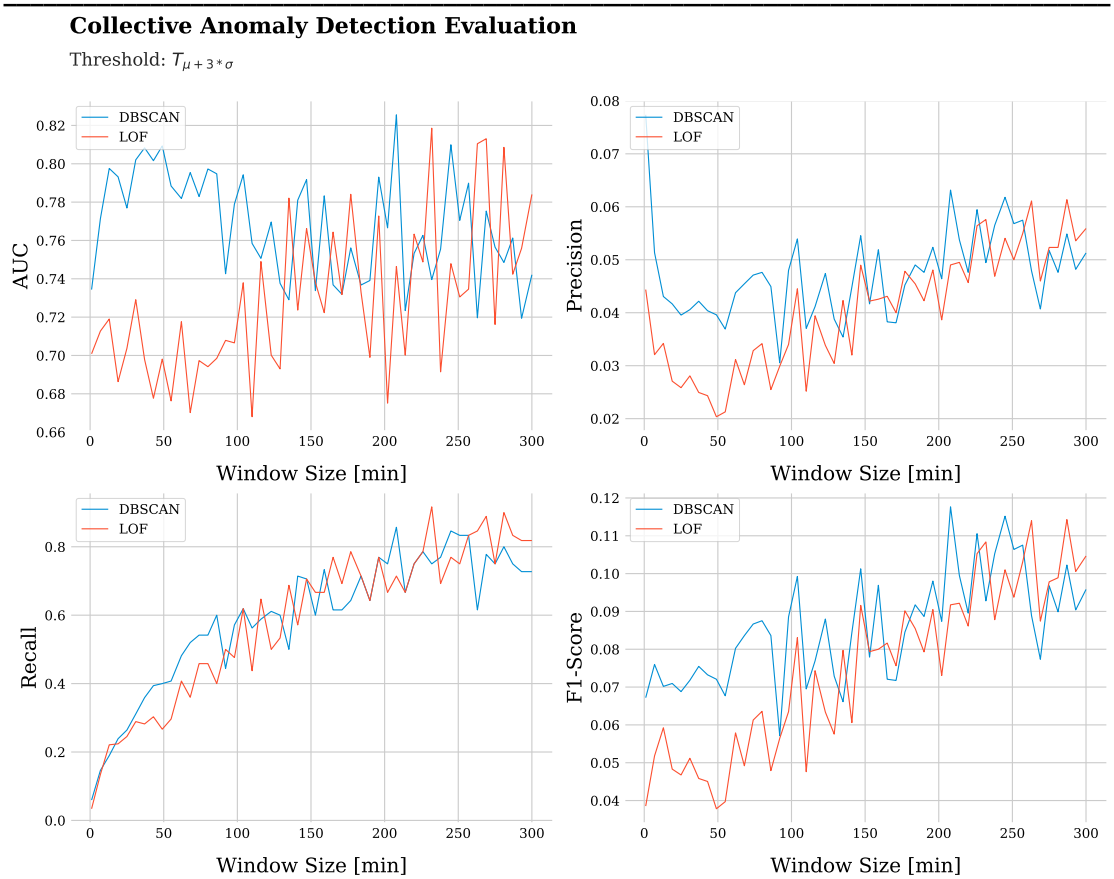


Figure 5.12.: Results of collective anomaly detection evaluation for models, threshold $T_{\mu+3*\sigma}$

we come up with a threefold evaluation approach to consider different aspects of quality of detection.

The label-based detection evaluation approach yields a naturally imbalanced ratio of FP to TP. Nevertheless, when comparing AUC and recall, the LOF model performs slightly better as its recall is higher for the threshold $T_{99\%}$.

> LOF model has minimal advantage over DBSCAN model when taking the label-based detection evaluation approach.

The point anomaly detection evaluation approach overcomes the problem of imbalance in classification metrics. However, by only considering point anomalies, the volatility of APM measures has a greater impact on the evaluation outcome. The results show

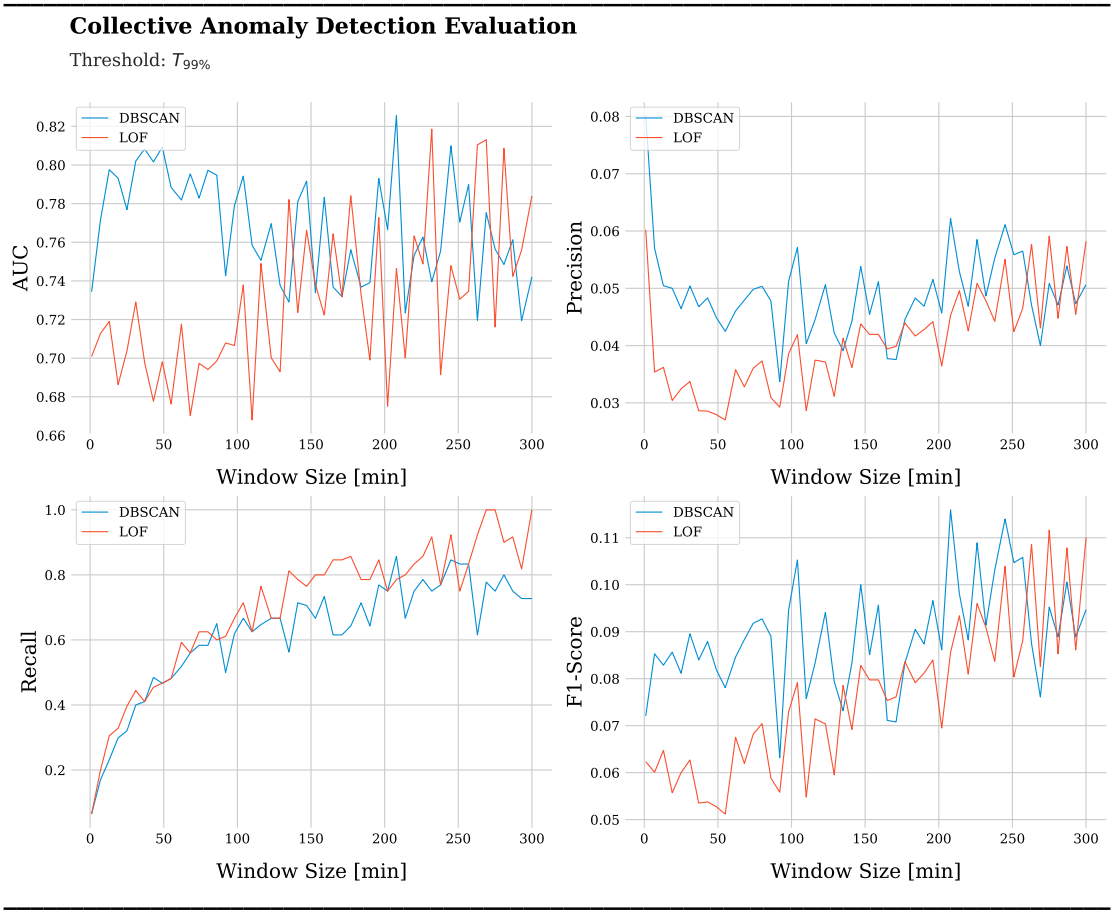


Figure 5.13.: Results of collective anomaly detection evaluation for models, threshold $T_{99\%}$

that the DBSCAN model outperforms the LOF model in all four considered evaluation metrics.

> The DBSCAN model is superior to the LOF model with respect to the point anomaly detection evaluation approach.

The collective anomaly detection evaluation approach addresses the problem of imbalance in classification metrics by sliding windows of varying size across the two months of data points. The volatility of APM measures has less impact as collective anomalies in the form of the windows are considered. By examining the four evaluation metrics plotted over different window sizes, we find that especially for smaller window sizes the DBSCAN model yields better results than the LOF model. For larger window sizes, the evaluation metrics are approximately on an equal level. However, for the largest window sizes ($>$ 250 minutes) the LOF model performs better than the DBSCAN model.

In total, the DBSCAN model has a better AUC for 39 of the 50 considered window sizes and is thus superior for our purposes, as AUC is our primary evaluation metric. The average AUC of the two models over all 50 window sizes is 0.77 for the DBSCAN model and 0.73 for the LOF model.

> Regarding the collective anomaly detection approach (sliding window), the DBSCAN model performs better than the LOF model in detecting abnormal system behavior.

Root Cause Analysis

RC_2 emphasizes the implementation of a model which not only detects anomalies, but also provides an indication for the root cause of abnormal system behavior.

As outlined in subsection 5.2.4, the DBSCAN model provides an indication for the root cause of a high anomaly index by considering individual contributions of the features to the total euclidean distance to the centroid of the normal cluster. In the developed software solution, this indication is given by the ratio of each individual distance to the total distance: a high percentage of the total distance indicates a high contribution to the abnormal system state. Figure 5.14 illustrates the UI for the indication of the root cause: by moving the cursor across the plotted anomaly index over time, the user can see the n relative distance contributions of the n selected features used for the clustering.

The LOF model, on the other hand, does not provide an indication for the root cause of its high anomaly index. Therefore, this is a significant shortcoming with respect to the previously defined Req_4 and RC_2.

As a result, only the DBSCAN model meets the requirements of the research artifact. However, it is noteworthy that the indication for the root cause which is provided by the DBSCAN model is still ought to be evaluated. This is further discussed in section 5.4.

> The only model implemented which meets the requirement of providing a root cause for the detected abnormal system behavior is the DBSCAN model.

Definition of Research Artifact

The definition of the research artifact as the best performing model for detection of abnormal system behavior in enterprise application is straightforward. Albeit the LOF is successfully applied in other fields of outlier detection, the developed approach of using DBSCAN for clustering and deriving an anomaly index yields better evaluation results (Goldstein and Uchida, 2016). As the previous two paragraphs show, the DBSCAN model outperforms the LOF model in two out of three evaluation approaches

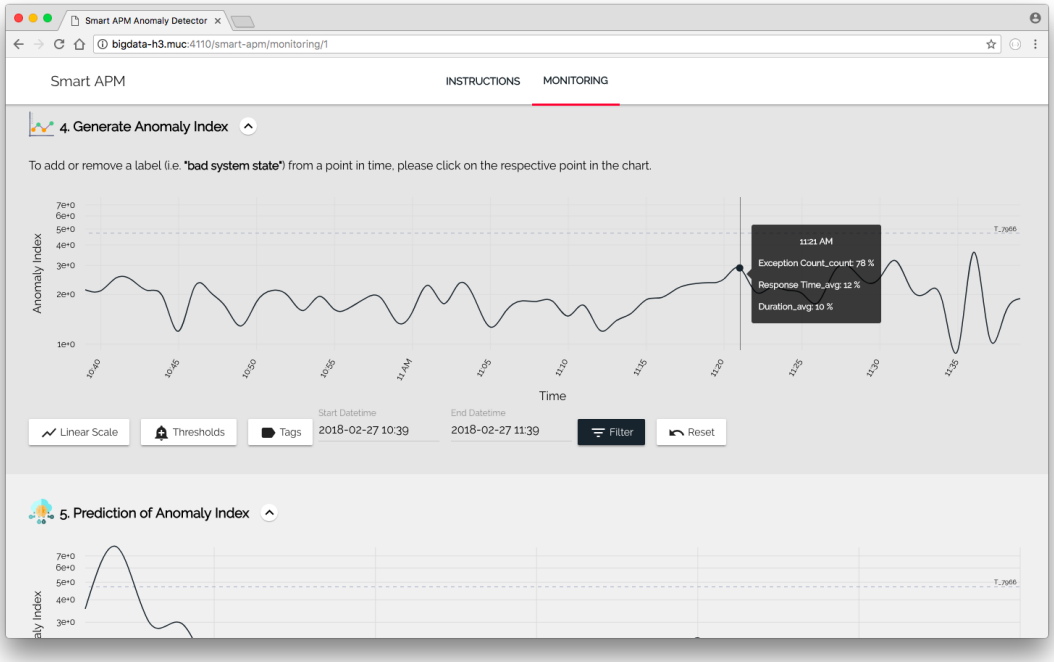


Figure 5.14.: UI for root cause indication from anomaly index

regarding its quality of detection and at the same time provides an additional indication for the root cause of the anomaly.

Concluding, we constitute the second research artifact as the implemented DBSCAN model with respect to RC_2.

> The second research artifact is a model based on DBSCAN which uses a multivariate set of features to derive an anomaly index reflecting the degree of abnormality in enterprise applications.

Summary

Table 5.5 summarizes the results of the evaluation.

5.4. Discussion & Conclusion

In this section, we discuss the evaluation results of RC_2 and RQ_3 and the limitations of the applied approach which are beyond the scope of this thesis.

Table 5.5.: Summary of evaluation results

	Results
1. Anomaly Detection Accuracy	• Label-Based Detection: LOF better than DBSCAN • Point Anomaly Detection: DBSCAN better than LOF • Collective Anomaly Detection: DBSCAN better than LOF
2. Root Cause Analysis	• DBSCAN provides indication for root cause through relative distances • LOF provides no indication for root cause
3. Definition of Research Artifact	• Best model implements DBSCAN to derive an anomaly index

First, RQ_3 is answered by looking at two implemented anomaly detection models which are compared through the aforementioned threefold evaluation. Generally, the label-based evaluation is the most intuitive approach. The primitive interpretation is that the models are able to detect between five and six of eight labeled anomalies. This raises the question whether these models are actually beneficial for deployment in real-world scenarios. Certainly, this recall of 0.625 to 0.75 is not perfect as the generated anomaly index does not reflect every abnormal system behavior. However, the models are not yet tuned, neither regarding their model hyperparameters, nor the used multivariate feature set. Hence, there is still a large potential to improve the models over time by collecting a more significant amount of labeled anomalies. Furthermore, as described in subsection 5.3.3, the label-based evaluation – even though being the most intuitive approach – is rather error-prone. Besides, the focus of this thesis is to prove the proficiency of the developed DBSCAN model by comparing it to a commonly used outlier detection method, the LOF, in the detection of abnormal system behavior in enterprise applications. Nevertheless, future work can make use of the insights of this thesis and compare the quality of detection of the DBSCAN model to existing APM software, e.g., Dynatrace.

Second, RC_2 constitutes that the implemented model provides an indication for the root cause of the detected abnormal system behavior. In subsection 5.3.4, we outline that only the DBSCAN model meets this requirement. Since large technology companies as Netflix are taking related approaches for root cause analysis, we emphasize the operational potential of our approach of an anomaly detection model which not only detects outliers, but provides context information (Fisher-Odgen et al., 2015). The evaluation of the indication of the root cause is still to be due and thus is a limitation of this thesis. Consequently, a rigorous evaluation requires a set of labeled anomalies with context information about the actual root cause of the abnormal system behavior.

Third, as mentioned in subsection 5.3.3, the evaluation approach we take induces a bias regarding the interpretation of partly detected abnormal system behaviors. Therefore,

one can elaborate on different strategies than interpreting a labeled anomalous time frame (or fixed time window) as completely detected if the maximal anomaly index exceeded a critical threshold. Other strategies are taking the mean, median, or a percentile of the distribution of the anomaly index within the considered time frame. Due to the limited scope of this thesis, addressing the problem of evaluation bias is encouraged in future research.

Fourth, in chapter 1, we emphasize a shift towards pro-active APM, whereas in chapter 4, we implement models to forecast the performance of enterprise applications. However, in this chapter, we mainly focus on retrospective modeling and detection of abnormal system states. Hence, for further research, we encourage a quantitative evaluation of a prediction of the developed anomaly index. Ultimately, this enables practitioners to detect abnormal system behavior beforehand and pro-actively solve the underlying technical problems. A naive approach is to create a univariate time series forecasting problem for the anomaly index, i.e., predict the anomaly index based on its historical values. The developed software solution uses a random forest regressor with univariate features from the anomaly index of the past 120 minutes, i.e., lag 120 (see subsection 4.2.3 and subsection 4.2.4 for further explanation), to forecast the next 30 minutes of the anomaly index. Figure 5.15 depicts the UI for the prediction of the anomaly index for the next 30 minutes. As aforementioned, the evaluation of the prediction of the anomaly index is beyond the scope of this thesis.

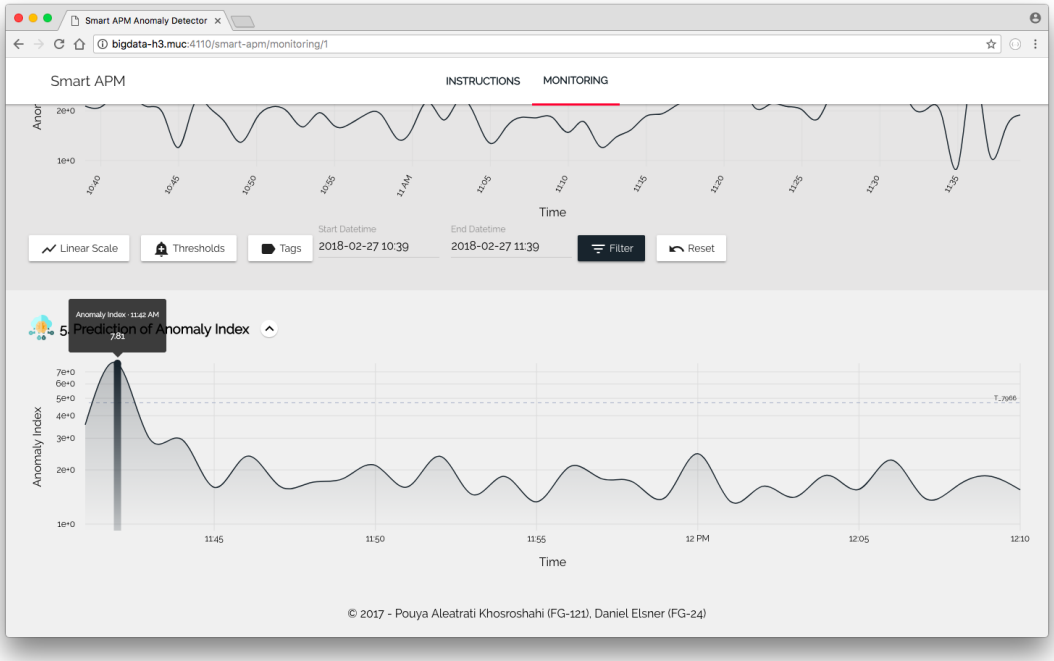


Figure 5.15.: UI for prediction of anomaly index

Fifth, the developed software solution is admittedly a lot more sophisticated than a general labeling tool. The software components encompass the following functionality:

1. Labeling time periods of abnormal system behavior
2. Manually selecting feature set (i.e., APM measures) for training of DBSCAN model
3. Continuous monitoring of the generated anomaly index
4. Root cause analysis through individual contributions of selected features to anomaly index
5. Manual and automatic threshold and notification management for the anomaly index
6. Prediction of the anomaly index in real-time (i.e., as soon as new monitoring data becomes available)

The software solution has yet been used to obtain eight labeled abnormal system behaviors, but can certainly be utilized for far more (monitoring) tasks due to its feature-richness. Generally, the underlying generic problem of performing anomaly or outlier detection on multivariate time series is transferable to several domains of application. For instance, the German car manufacturer, which provides the APM measures for our analyses, plans to use the software for detecting anomalies in time series collected from production robots. As a result, the software solution itself is ought to be evaluated qualitatively in further examinations and research.

Sixth, we lay stress on the dependence of the anomaly detection model evaluation and the quality of obtained labels for abnormal system behavior. Naturally, the labels which are collected from practitioners can be heterogeneous and driven by subjective judgment and estimation. There are other proxies which can be considered for anomalies in enterprise applications, e.g., high response time or amount of HTTP errors as defined in SLAs (see subsection 2.1.3). However, it is reasonable to rely on the manually collected labels for evaluating the implemented models, as they are the only ground truth for abnormal system behavior which is specifically available for the examined enterprise application.

Seventh, as in chapter 4, the available real-world monitoring data defines the scope for the analyses in section 5.2. The developed models are thus evaluated and built on fifteen APM measures – as time series data – from one enterprise application. Certainly, this makes an impact on the generalizability of the findings within this thesis. Further research could evaluate the implemented models on more enterprise applications in different organizations, and harness other APM data sources for the model building. Another approach for detecting anomalies might involve utilizing trace data, which holds less aggregated information at the price of intricacy and large volume of data.

Last, to approach the motivated change in paradigm towards proactive APM, we develop and evaluate models for detecting abnormal system behavior in enterprise applications.

However, the potentially generated business value, e.g., greater revenues through cost savings or higher customer satisfaction, is yet to be evaluated. The conducted analyses can hence serve as a starting point for further investigation regarding the monetary business impact of proactive APM. Consequently, in future research, derived insights could exploit the aforementioned optimization potential in EAM, i.e., improving information flows between EA layers, and thereby increase the alignment of business and IT.

Abridged Answer to RQ_3

The most accurate implemented **anomaly detection model is based on DBSCAN**, a density-based clustering algorithm. As in the present case, there are no labeled anomalies available ex ante, we **obtain eight labeled anomalies** in enterprise applications from APM practitioners. These serve as the ground truth to assess the detection quality of the models. Overall, throughout a **threefold evaluation process**, we find that the **DBSCAN model detects abnormal system behavior more reliable than the commonly used outlier detection technique LOF** which we consider as a reference model. Notable is the **indication for the root cause** of the detected abnormal system behavior that the DBSCAN model additionally outputs.

Quantitative Results for Best Model: DBSCAN

1. Label-Based Evaluation
2. Point Anomaly Detection Evaluation
3. Collective Anomaly Detection Evaluation (average AUC over 50 window sizes between 1 and 300 min)

	AUC_1 $[\frac{\%}{100}]$	AUC_2 $[\frac{\%}{100}]$	AUC_3 $[\frac{\%}{100}]$
App_1	0.97	0.73	0.77

6. Conclusion

In this chapter, we recap the research methodology and summarize the major findings of this thesis. Moreover, we discuss the capability and relevance of productive deployment of the developed research artifacts. Finally, limitations attributable to the scope of this thesis are pointed out and an outlook on possible further research is provided.

6.1. Summary & Discussion

We start by motivating a shift towards proactive and analytical APM to increase the business value of enterprise applications in chapter 1. The research topic is further narrowed down with respect to two main research goals, (i) predicting enterprise application performance, and (ii) detection of abnormal system behavior in enterprise applications. Both of these are reflected in the overarching RQ_1, which intends to outline existing research gaps. Furthermore, $RQ_{2.1}$ and $RQ_{2.2}$ address (i) and RQ_3 covers (ii). In chapter 2, terminologies of APM, EAM, and ML are introduced to constitute substantial foundations and to provide insights into applied methodologies within this thesis. According to RQ_1, in chapter 3, we conduct a literature review yielding two RCs which emphasize the application of ML techniques in the domain of APM. Consequently, we proceed with the development of these RCs.

Completion RQ_1

RQ_1: What are research gaps in existing scientific work regarding enterprise application performance forecasting and detection of abnormal system behavior in enterprise applications?

The identified research gaps are:

- **Enterprise Application Performance Forecasting:** Investigation of **linear and nonlinear** ML techniques for **multivariate multi-step** time series performance forecasting.
- **Detection of Abnormal System Behavior:** Investigation of **multivariate density-based anomaly detection** techniques providing an **indication for the root cause** of abnormal system behavior.

6.1.1. Predicting Enterprise Application Performance

Besides the identification of existing research gaps, chapter 3 reveals response time as the most relevant APM measure to determine performance of enterprise applications in prevalent research. This answers $RQ_{2.1}$ and therefore, in chapter 4, we elaborate on models for multi-step time series forecasting of the response time in enterprise applications.

Completion $RQ_{2.1}$

$RQ_{2.1}$: Which are relevant APM measures to assess enterprise application performance?

From the reviewed related scientific (and non-scientific, see subsection 2.1.3) work, **response time** can be identified as the **most relevant APM measure** which reflects the performance of an enterprise application.

$RQ_{2.2}$ and RC_1 further postulate a comparison of linear and nonlinear regression techniques from supervised ML on uni- as well as multivariate feature sets. We delineate the process of data identification and preparation, feature engineering and selection, and model implementation. The data set for the analyses contains fifteen APM measures for two enterprise applications and is obtained from three APM source systems of a German car manufacturer. The implemented models comprise a baseline, linear regression, random forest regression, MLP, and LSTM model, which all use uni- and multivariate feature sets. Their predictive accuracy is compared regarding a forecasting horizon of 30 minutes of the response time.

The quantitative evaluation in section 4.3 yields the first research artifact: The random forest regression model on a multivariate feature set is the most accurate among the implemented models for forecasting the response time 30 minutes beforehand. However, the evaluation shows that extreme values are difficult to predict due to randomness and volatility of the investigated APM measures. But these are of particular interest in practice as high peaks might reflect performance issues in the enterprise application.

The capability of productive deployment of the developed research artifact is ambiguous: Throughout the analysis, we find that the process of feature engineering and model tuning requires application-specific knowledge and indispensable manual efforts. Hence, it is difficult to generalize the prediction models to suit an arbitrary enterprise application. Furthermore, considering and predicting a single APM measure can be insufficient due to the heterogeneous character of applications. During the operation of one enterprise application, peaks in the response time might reflect performance issues. However, in another one they might be distinctive as of the used programming patterns (i.e., (a)synchronous programming) in the source code. Concluding, even though a productive deployment would certainly require further application-specific investigations, it may be reasonable for business-critical enterprise applications. The transferability of

the predictive model to other enterprise applications is yet questionable since specific analyses are required.

Completion $RQ_{2.2}$

$RQ_{2.2}$: How well can the response time of an enterprise application be predicted based on historical APM measures?

The analyses on forecasting the response time in enterprise applications based on historical APM measures yield that – in the present case – **random forest regressors** constitute the most accurate prediction models. The best predictions are obtained by harnessing the information of multiple APM measures through **multivariate modeling**. **Application-specific** considerations and knowledge are beneficial, if not indispensable, for feature engineering and selection in the process of model building and refinement. However, the models have **deficiencies** regarding the **prediction of extreme observations** and the accuracy decreases with **increasing forecasting horizon**.

Quantitative Results for Best Model: Random Forest Regressor (multivariate)

	MAE $[ms]$	MAPE $[\frac{\%}{100}]$	RMSE $[ms]$	RMSLE $[log(ms)]$	RMSPE $[\frac{\%}{100}]$
App_1	196.97	0.23	398.02	0.30	0.70
App_2	163.17	0.31	269.86	0.37	0.40

6.1.2. Anomaly Detection in Application Performance Management

In chapter 5, we elaborate on the development of multivariate anomaly detection models with respect to RQ_3 and RC_2. After deliberate consideration of existing approaches from practitioners (e.g., Netflix, Fisher-Odgen et al., 2015), we emphasize to use the two unsupervised ML techniques DBSCAN and LOF for building anomaly detection models. Furthermore, in order to evaluate these two models, we design and implement a software solution which enables obtaining anomalous data labels from practitioners. The source systems and the data set for the model implementations are equal to chapter 4, but we are limited to one enterprise application. Both models result in a so-called anomaly index, which is a continuous measure reflecting the degree of abnormality of the enterprise application over time. However, the implemented DBSCAN model is a novel concept in comparison to the commonly used outlier detection technique LOF. In addition to the anomaly index, the DBSCAN model provides an indication for the root cause of the detected anomaly.

Regarding the appraisal of the expressiveness of this anomaly index, we illuminate the difficulty of the evaluation process of anomaly detection models. As a consequence, we derive a threefold evaluation approach, which considers possible biases in different

evaluation methodologies. The evaluation yields that based on eight manually labeled abnormal system behaviors the DBSCAN model is more accurate in detecting anomalies in enterprise applications than the LOF model. Moreover, the DBSCAN model offers an indicator for root cause analysis and is thus defined as the second research artifact capturing the essence of RQ_3.

The capability of productive deployment of the developed research artifact is proven through the already provided software solution, which encloses the DBSCAN model. The feedback from domain experts confirms that the developed solution is a more transparent and intelligible approach for anomaly detection and root cause analysis than professional APM software, e.g., Dynatrace. Regarding the quality of detection of the research artifact, the DBSCAN model – in the present case – performs better than the common LOF outlier detection technique, which endorses its application.

Completion RQ_3

RQ_3: How well can abnormal system behavior of enterprise applications be detected based on APM measures?

The most accurate implemented **anomaly detection model is based on DBSCAN**, a density-based clustering algorithm. As in the present case, there are no labeled anomalies available ex ante, we **obtain eight labeled anomalies** in enterprise applications from APM practitioners. These serve as the ground truth to assess the detection quality of the models. Overall, throughout a **threefold evaluation process**, we find that the **DBSCAN model detects abnormal system behavior more reliable than the commonly used outlier detection technique LOF** which we consider as a reference model. Notable is the **indication for the root cause** of the detected abnormal system behavior that the DBSCAN model additionally outputs.

Quantitative Results for Best Model: DBSCAN

1. Label-Based Evaluation
2. Point Anomaly Detection Evaluation
3. Collective Anomaly Detection Evaluation (average AUC over 50 window sizes between 1 and 300 min)

	AUC_1 $[\frac{\%}{100}]$	AUC_2 $[\frac{\%}{100}]$	AUC_3 $[\frac{\%}{100}]$
App_1	0.97	0.73	0.77

6.2. Limitations & Outlook

As aforementioned, we constitute two research artifacts in (i) predicting enterprise application performance, and (ii) anomaly detection in enterprise applications. Due to

the scope of this thesis, there exist a number of limitations and consequently levers in which we see the potential for future research.

The developed research artifacts approach the motivated change in paradigm towards proactive APM. However, the potentially generated business value, e.g., greater revenues through cost savings or higher customer satisfaction, is yet to be evaluated. The conducted analyses can hence serve as a starting point for further investigation regarding the monetary business impact of proactive APM. Consequently, in future research, derived insights could exploit optimization potential in EAM, e.g., enhance application landscape or leverage synergies in EA (see section 2.2), and thereby increase the alignment of business and IT.

Within the prediction of performance of enterprise applications, we are limited regarding model tuning and feature engineering. Further endeavors could build on top of the modest MLP and LSTM architectures or put emphasis on different modeling techniques. Moreover, there is a variety of unsupervised ML techniques for feature engineering, e.g., PCA or NMF, which might unveil meaningful latent variables for potential features. We suggest using the developed predictive models on other related benchmark data sets, to get a better impression of their predictive accuracy. In addition to that, future research could evaluate the models on more enterprise applications in different organizations, and harness other APM data sources for the model building.

In detecting abnormal system behavior in enterprise applications, we encounter limitations regarding the evaluation of the root cause indicator provided by the research artifact. Future research could utilize context information about labeled abnormal system behavior for an in-depth evaluation of the root cause indication. Aside from that, the scope of the evaluation of the anomaly detection models could be extended to a quantitative comparison against the anomaly detection components of existing APM software. We outline the obstacles and biases in evaluating the accuracy of anomaly detection models, which we address with a threefold approach. The impact of these induced evaluation biases could further be discussed by focusing on other evaluation aspects. In section 5.4, we motivate the thorough development and evaluation of predictive models for the derived anomaly index. Hereby, the concept of the anomaly index would most certainly be enhanced since monitoring practitioners typically desire ex ante alerting or notification mechanisms. Finally, the statistical relevance of the anomaly detection models' evaluation could further be increased through a larger (ideally publicly available) data set with universally valid (i.e., less affected by subjective judgment) labeled anomalies. The availability of this labeled data would allow both, to consider other techniques for anomaly detection, e.g., supervised ML, and to compete with other public anomaly detection models.

A. Appendix

A.1. Predicting Enterprise Application Performance

A.1.1. Feature Engineering for App_2

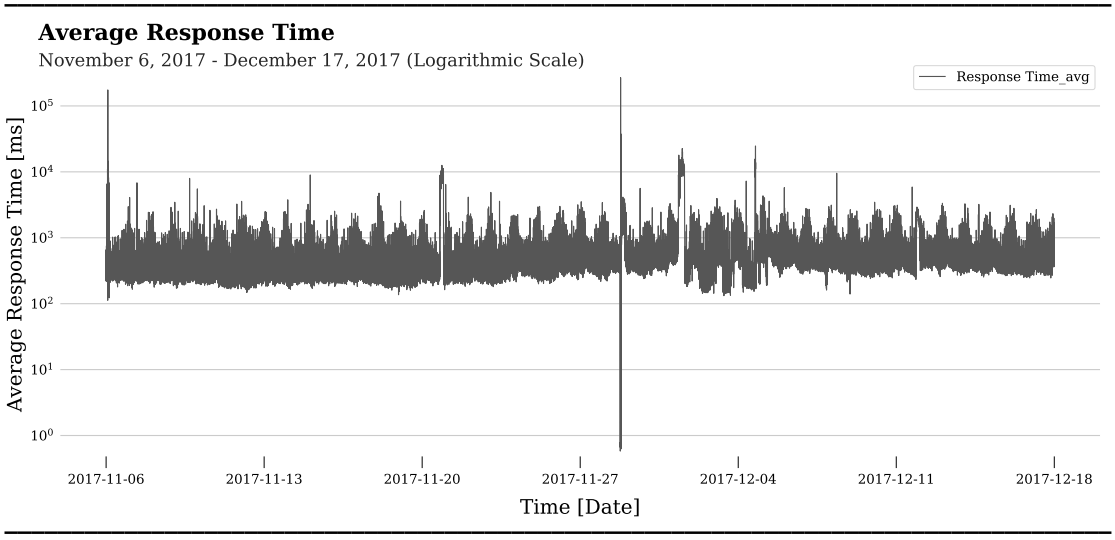


Figure A.1.: Logarithmic plot of historical ART of App_2

A.1.2. Model Implementations for App_2

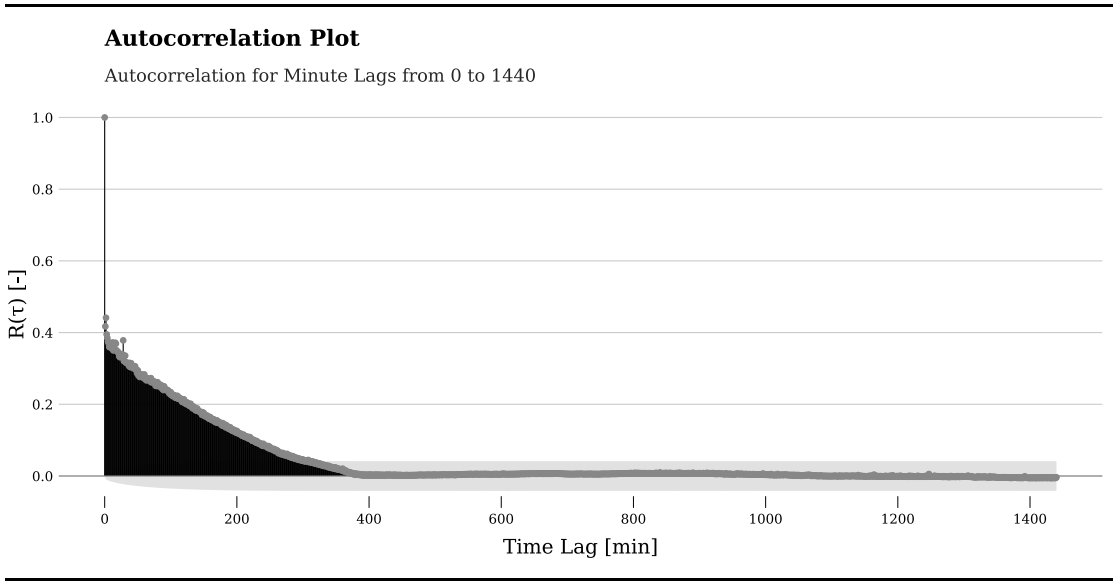


Figure A.2.: Autocorrelation plot of historical ART of App_2

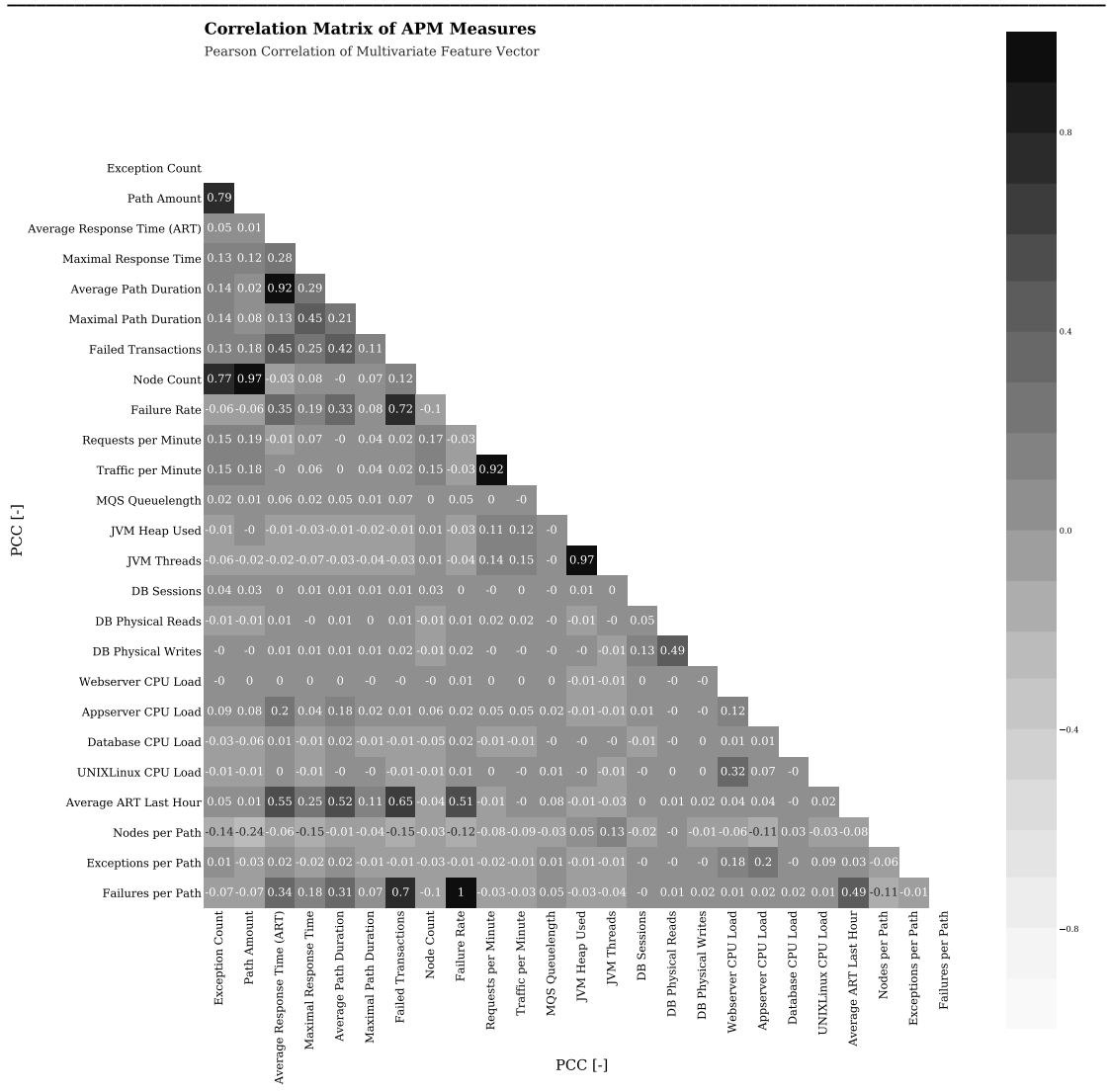


Figure A.3.: Correlation matrix of APM measures for App_2

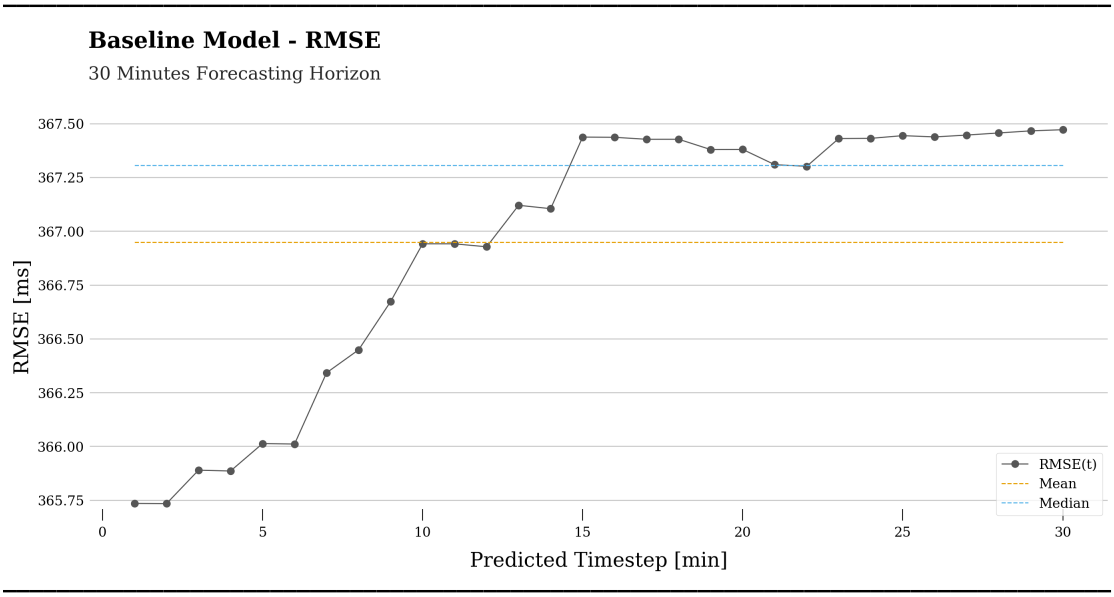


Figure A.4.: RMSE baseline model for App_2

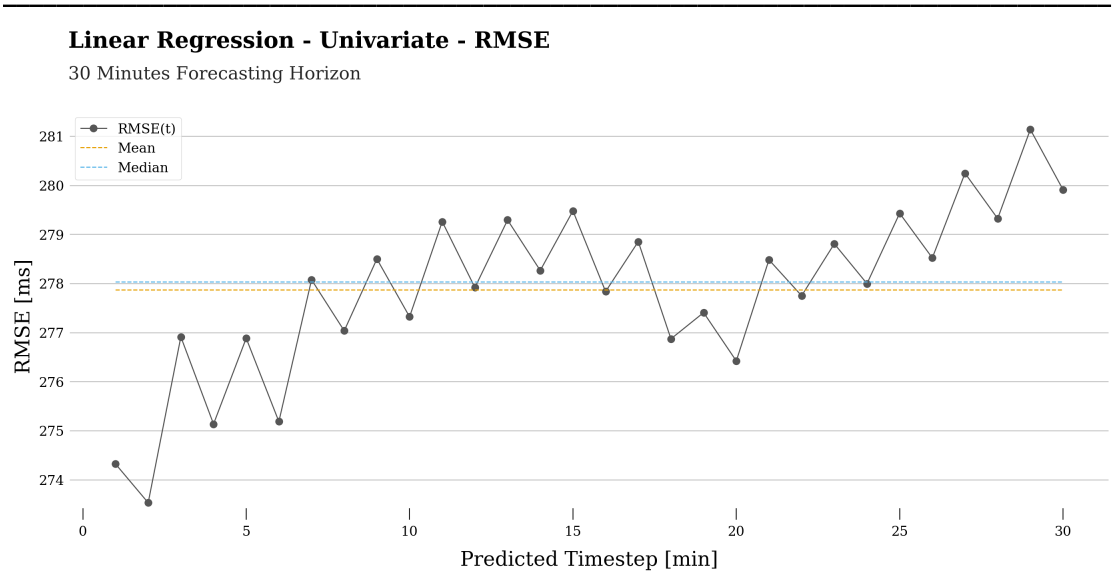


Figure A.5.: RMSE linear regression model, univariate feature set for App_2

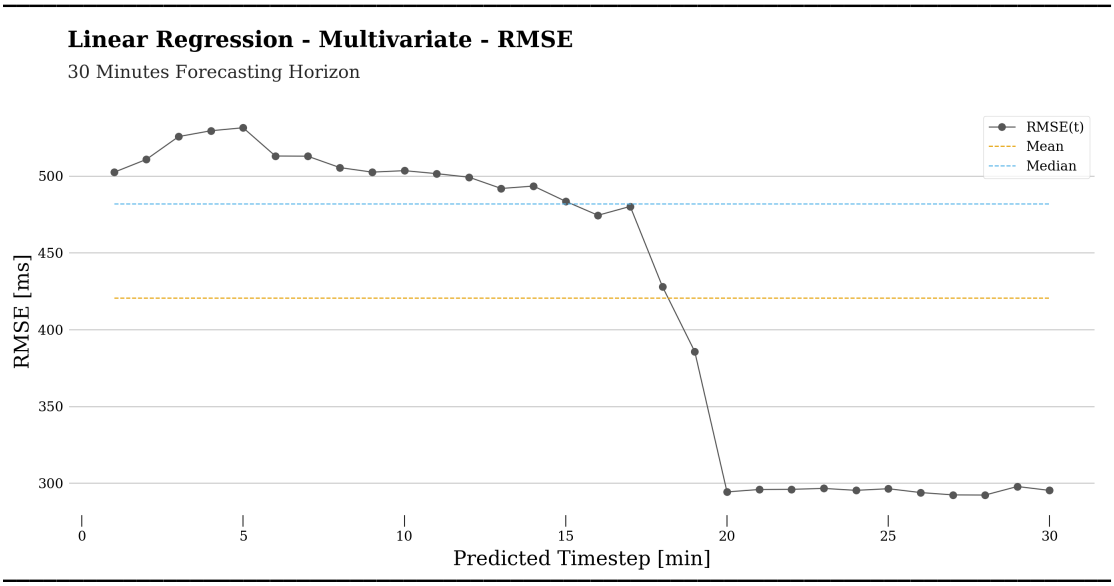


Figure A.6.: RMSE linear regression model, multivariate feature set for App_2

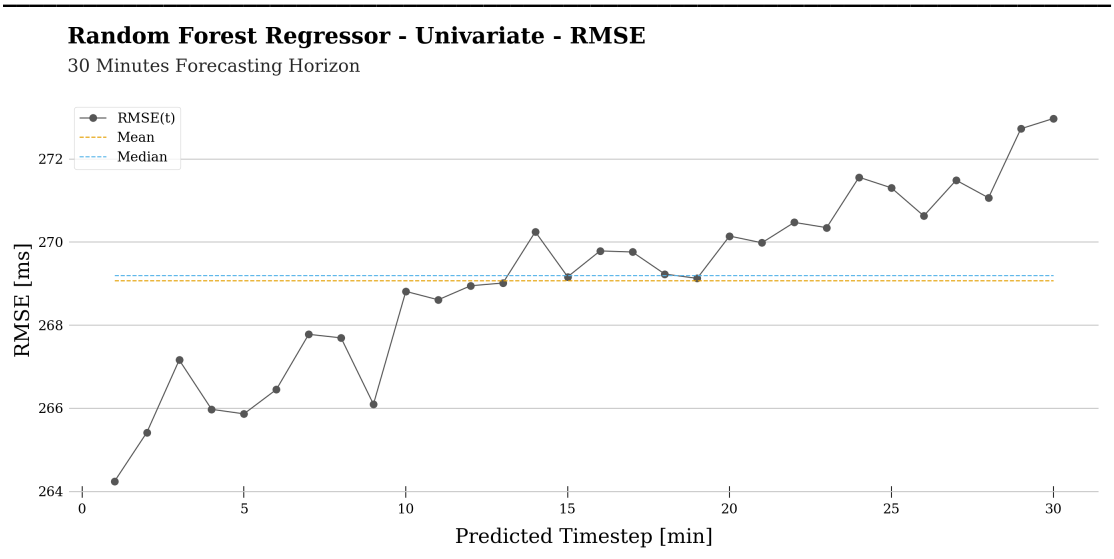


Figure A.7.: RMSE random forest regression model, univariate feature set for App_2

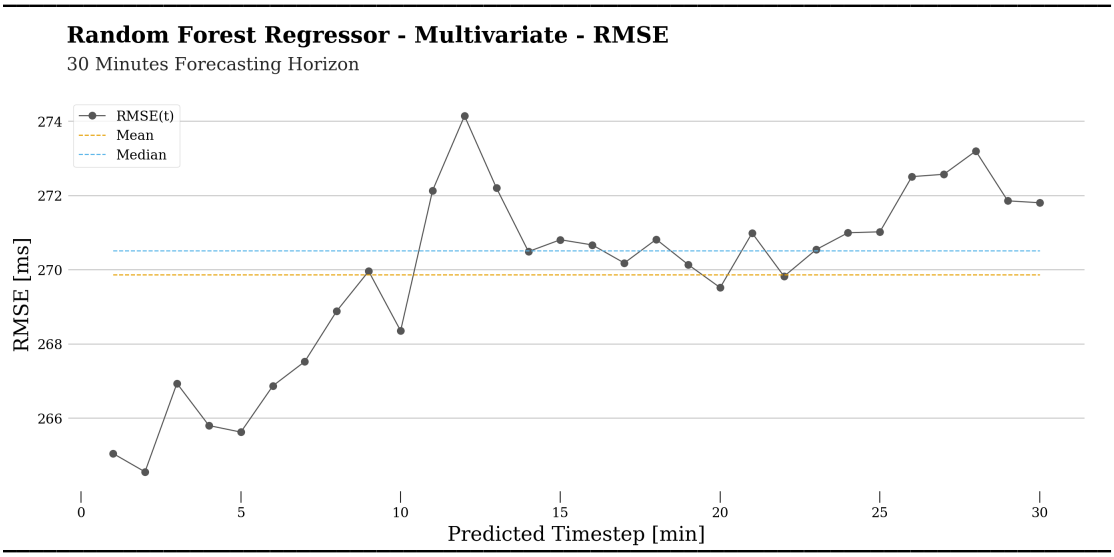


Figure A.8.: RMSE random forest regression model, multivariate feature set for App_2

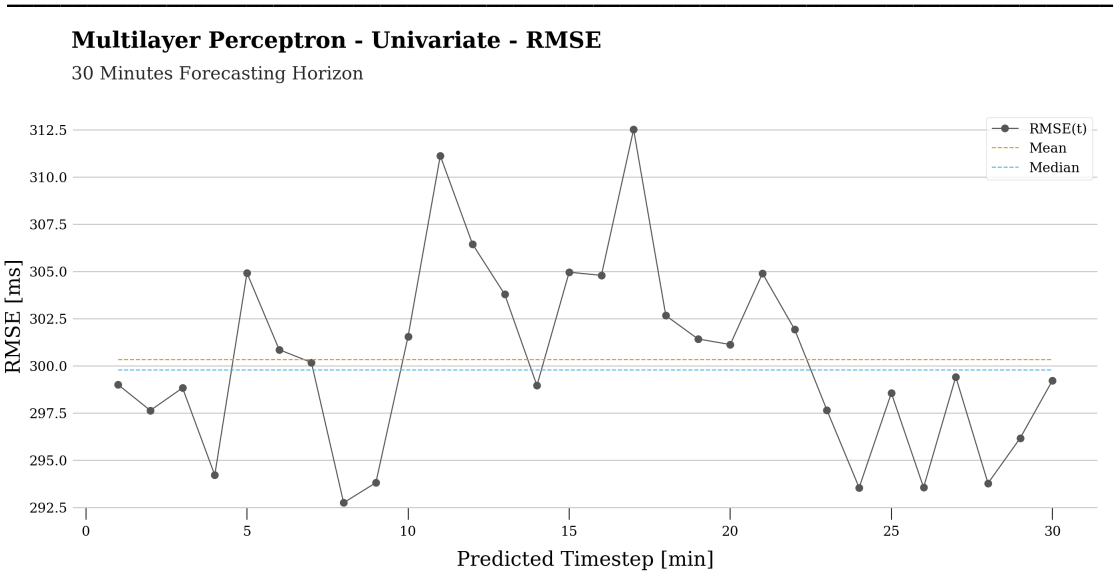


Figure A.9.: RMSE MLP regression model, univariate feature set for App_2

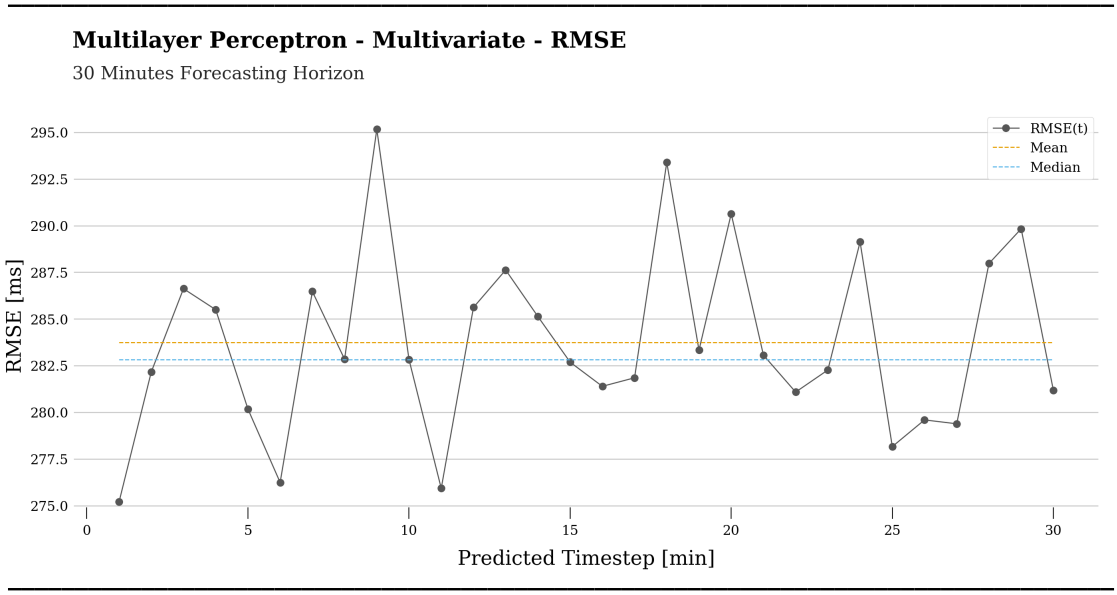


Figure A.10.: RMSE MLP regression model, multivariate feature set for App_2

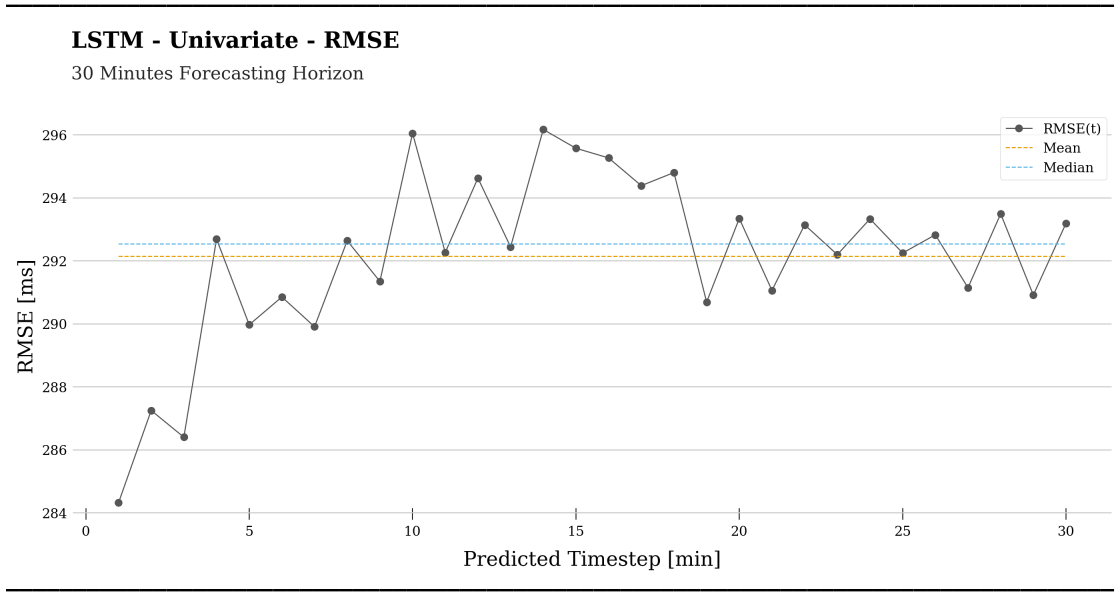


Figure A.11.: RMSE LSTM regression model, univariate feature set for App_2

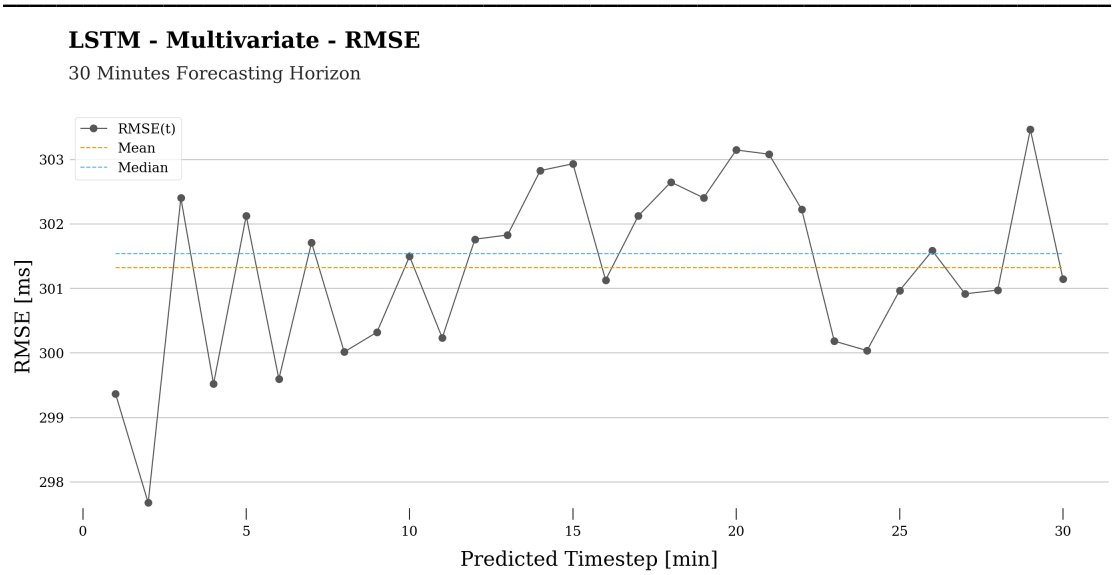


Figure A.12.: RMSE LSTM regression model, multivariate feature set for App_2

Bibliography

Aboagye-Sarfo, P., Q. Mai, F. M. Sanfilippo, D. B. Preen, L. M. Stewart, and D. M. Fatovich (2015). "A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in Western Australia." In: *Journal of Biomedical Informatics* 57, pp. 62–73.

Agyemang, M., K. Barker, and R. Alhajj (2006). "A comprehensive survey of numeric and symbolic outlier mining techniques." In: *Intelligent Data Analysis* 10.6, pp. 521–538.

Ahmad, S., A. Lavin, S. Purdy, and Z. Agha (2017). "Unsupervised real-time anomaly detection for streaming data." In: *Neurocomputing* 262, pp. 134–147.

Ahmed, N. K., A. F. Atiya, N. El Gayar, and H. El-Shishiny (2010). "An empirical comparison of machine learning models for time series forecasting." In: *Econometric Reviews* 29.5, pp. 594–621.

Ahmed, T. M., C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang (2016). "Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications." In: *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*, pp. 1–12.

Aier, S., B. Gleichauf, and R. Winter (2011). "Understanding enterprise architecture management design - an empirical analysis." In: *International Conference on Wirtschaftsinformatik*. Vol. 10. February, pp. 645–654.

Albu, R.-D., I Felea, and F Popentiu-Vladicescu (2013). "On the best adaptive model for web services response time prediction." In: *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 39–42.

Aleskerov, E., B. Freisleben, and B. Rao (1997). "CARDWATCH: A neural network based database mining system for credit card fraud detection." In: *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pp. 220–226.

Amani, P., M. Kihl, and A. Robertsson (2011). "NARX-based multi-step ahead response time prediction for database servers." In: *International Conference on Intelligent Systems Design and Applications, ISDA*, pp. 813–818.

Ara, A., F. Louzada, and C. A. Diniz (2017). "Statistical monitoring of a web server for error rates: a bivariate time-series copula-based modeling approach." In: *Journal of Applied Statistics* 44.13, pp. 2287–2300.

Baraglia, R and P Palmerini (2002). "SUGGEST: A web usage mining system." In: *Proceedings - International Conference on Information Technology: Coding and Computing, ITCC 2002*, pp. 282–287.

Bell, S. (2015). *6 critical web application performance metrics to consider*. URL: https://www.cdnetworks.com/en/news/6-critical-web-application-performance-metrics-to-consider/4257 (visited on 03/29/2018).

Ben Taieb, S., A. Sorjamaa, and G. Bontempi (2010). "Multiple-output modeling for multi-step-ahead time series forecasting." In: *Neurocomputing* 73.10-12, pp. 1950–1957.

Ben Taieb, S., G. Bontempi, A. F. Atiya, and A. Sorjamaa (2012). "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition." In: *Expert Systems with Applications* 39.8, pp. 7067–7083.

Bharadwaj, A., O. A. El Sawy, P. A. Pavlou, and N. Venkatraman (2013). "Digital business strategy: toward a next generation of insights." In: *MIS Quarterly* 37.2, pp. 471–482.

Bontempi, G. (2008). "Long term time series prediction with multi-input multi-output local learning." In: *Proceedings of the 2nd European symposium on time series prediction*. 1, pp. 145–154.

Bontempi, G., S. Ben Taieb, and Y. A. Le Borgne (2013). "Machine learning strategies for time series forecasting." In: *Lecture Notes in Business Information Processing*. Vol. 1, pp. 62–77.

Borzemski, L. and A. Kamińska-Chuchmała (2011a). "3D web performance forecasting using turning bands method." In: *International Conference on Computer Networks*. Ed. by A. Kwiecień, P. Gaj, and P. Stera. Vol. 160. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 102–113.

– (2011b). "Knowledge discovery about web performance with geostatistical turning bands method." In: *Lecture Notes in Computer Science* 6882.2, pp. 581–590.

– (2013). "Spatial econometrics models in web server's performance." In: *Communications in Computer and Information Science* 370, pp. 45–54.

– (2014). "Web server's performance prediction with using spatial econometric methods." In: *Rynek Energii* 112.3, pp. 120–124.

– (2016). "Distributed web server's data performance processing with application of spatial econometrics models." In: *Advances in Intelligent Systems and Computing* 430, pp. 37–48.

Braun, C. and R. Winter (2005). "A comprehensive enterprise architecture metamodel and its implementation using a metamodeling platform." In: *Enterprise Modelling and Information Systems Architectures, Proc. of the Workshop in Klagenfurt, GI-Edition Lecture Notes (LNI)*, pp. 64–79.

Breiman, L, J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and regression trees*. Vol. 19, p. 368.

Breiman, L. (2001). "Random forests." In: *Machine Learning* 45.1, pp. 5–32.

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander (2000). "LOF: Identifying density-based local outliers." In: *Proceedings of the 2000 Acm Sigmod International Conference on Management of Data*, pp. 93–104.

Brunnert, A. and H. Krcmar (2015). "Detecting performance change in enterprise application versions using resource profiles." In: *EAI Endorsed Transactions on Self-Adaptive Systems* 1.3 (e4), pp. 165–172.

Buckl, S. (2011). "Developing organization-specific enterprise architecture management functions using a method base." PhD thesis, p. 344.

Buckl, S., F. Matthes, and C. M. Schweda (2011). "Socio-technic dependency and rationale models for the enterprise architecture management function." In: *Advanced Information Systems Engineering Workshops*, pp. 528–540.

Campello, R. J.G. B., D. Moulavi, and J. Sander (2013). "Density-based clustering based on hierarchical density estimates." In: *Advances in Knowledge Discovery and Data Mining*. Vol. 7819, pp. 160–172.

Casdagli, M., S. Eubank, J. D. Farmer, and J. Gibson (1991). "State space reconstruction in the presence of noise." In: *Physica D: Nonlinear Phenomena* 51.1-3, pp. 52–98.

Chandola, V., A. Banerjee, and V. Kumar (2009). "Anomaly detection." In: *ACM Computing Surveys* 41.3, pp. 1–58.

Cheng, H., P. Tan, J. Gao, and J. Scripps (2006). "Multistep-ahead time series prediction." In: *Advances in Knowledge Discovery and Data Mining*, pp. 765–774.

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734.

Cunha, C. A. and L. Moura E Silva (2012). "Separating performance anomalies from workload-explained failures in streaming servers." In: *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, pp. 292–299.

– (2013). "Prediction of performance failures in video-streaming servers." In: *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, pp. 283–292.

De Gooijer, J. G. and R. J. Hyndman (2006). "25 years of time series forecasting." In: *International Journal of Forecasting* 22.3, pp. 443–473.

Din, G. M. U. and A. K. Marnerides (2017). "Short term power load forecasting using Deep Neural Networks." In: *2017 International Conference on Computing, Networking and Communications (ICNC)*, pp. 594–598.

Djuriš, J., D. Medarević, M. Krstić, I. Vasiljević, I. Mašić, and S. Ibrić (2012). "Design space approach in optimization of fluid bed granulation and tablets compression process." In: *The Scientific World Journal*.

Ester, M., H. P. Kriegel, J. Sander, and X. Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 226–231.

Feng, W and K Vij (2007). "Machine learning prediction and web access modeling." In: *Proceedings - International Computer Software and Applications Conference*. Vol. 2, pp. 607–612.

Fisher-Odgen, P., G. Burrell, C. Sanden, and C. Rioux (2015). *Tracking down the villains: outlier detection at Netflix*. URL: `https://medium.com/netflix-techblog/tracking-down-the-villains-outlier-detection-at-netflix-40360b31732` (visited on 03/16/2018).

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., p. 560.

Fujimaki, R., T. Yairi, and K. Machida (2005). "An approach to spacecraft anomaly detection problem using kernel feature space." In: *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 401–411.

Gardner, M. and S. Dorling (1998). "Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences." In: *Atmospheric Environment* 32.14-15, pp. 2627–2636.

Goel, H., I. Melnyk, and A. Banerjee (2017). "R2N2: Residual recurrent neural networks for multivariate time series forecasting." In: *CoRR* abs/1709.0.

Goldberg, K. (2017). *The most important metrics you need to monitor your application*. URL: `https://blog.appoptics.com/important-metrics-need-monitor-application/` (visited on 03/29/2018).

Goldstein, M. and S. Uchida (2016). "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data." In: *PLoS ONE* 11.4.

Graves, A, A.-R. Mohamed, and G Hinton (2013). "Speech recognition with deep recurrent neural networks." In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 6, pp. 6645–6649.

Guo, X, Z Zhao, and P Guo (2011). "Studies about convex hull support vector machine in the server performance alarm." In: *2011 3rd International Workshop on Intelligent Systems and Applications, ISA 2011 - Proceedings*.

Haight, C., W. Cappelli, and F. De Silva (2015). "Magic quadrant for application performance monitoring suites." In: *Gartner*.

Hamzaçebi, C., D. Akay, and F. Kutay (2009). "Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting." In: *Expert Systems with Applications* 36.2 (2), pp. 3839–3844.

Hasan, U. (2016). *How to use machine learning to debug outages 10x faster*. URL: `https://engr.overseerlabs.io/how-to-use-machine-learning-to-debug-outages-10x-faster-e19a97946f80` (visited on 03/16/2018).

– (2017). *Our challenges in building a "useful" anomaly detection system*. URL: `https://engr.overseerlabs.io/our-challenges-in-building-a-useful-anomaly-detection-system-1d589de77e60` (visited on 03/16/2018).

Hastie, T., R. Tibshirani, and J. Friedman (2001). "The elements of statistical learning." In: *The Mathematical Intelligencer* 27.2, pp. 83–85.

Heger, C., A. V. Hoorn, D. Okanovic, S. Siegl, and A. Wert (2016). "Expert-guided automatic diagnosis of performance problems in enterprise applications." In: *Proceedings - 2016 12th European Dependable Computing Conference, EDCC 2016*, pp. 185–188.

Heger, C., A. van Hoorn, M. Mann, and D. Okanović (2017). "Application performance management: State of the art and challenges for the future." In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17*, pp. 429–432.

Hevner, A. R., S. T. March, J. Park, and S. Ram (2004). "Design science in information systems research." In: *MIS Quarterly* 28.1, pp. 75–105.

Hochreiter, S. and J. J. Schmidhuber (1997). "Long short-term memory." In: *Neural Computation* 9.8, pp. 1–32.

Hodge, V. J. and J. Austin (2004). "A survey of outlier detection methodologies." In: *Artificial Intelligence Review* 22.2, pp. 85–126.

Hoffmann, G. A., K. S. Trivedi, and M. Malek (2007). "A best practice guide to resource forecasting for computing systems." In: *IEEE Transactions on Reliability* 56.4, pp. 615–628.

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558.

Hussain, T, S Asghar, and S Fong (2010). "A hierarchical cluster based preprocessing methodology for web usage mining." In: *Proc. - 6th Intl. Conference on Advanced Information Management and Service, IMS2010, with ICMIA2010 - 2nd International Conference on Data Mining and Intelligent Information Technology Applications*, pp. 472–477.

Hyndman, R. J. (1996). "Computing and graphing highest density regions." In: *American Statistician* 50.2, pp. 120–126.

Hyndman, R. J., E. Wang, and N. Laptev (2016). "Large-scale unusual time series detection." In: *Proceedings - 15th IEEE International Conference on Data Mining Workshop, ICDMW 2015*, pp. 1616–1619.

IEEE-SA Standards Board (2000). "IEEE recommended practice for architectural description of software-intensive systems." In: *IEEE Std* 1471-2000, pp. 1–23.

James, G., D. Witten, R. Tibshirani, and T. Hastie (2013). *An introduction to statistical learning with applications in R*, p. 431.

Jiang, N., R. Villafane, K. Hua, A. Sawant, and A. Prabhakara (2005). "ADMiRe: An algebraic data mining approach to system performance analysis." In: *IEEE Transactions on Knowledge and Data Engineering* 17.7, pp. 888–901.

Jiawei, H, M. Kamber, J. Han, M. Kamber, and J. Pei (2012). *Data mining: Concepts and techniques*, p. 745.

Kaisler, S. H., F. Armour, and M. Valivullah (2005). "Enterprise architecting: Critical problems." In: *Proceedings of the Annual Hawaii International Conference on System Sciences*, p. 224.

Kingma, D. P. and J. L. Ba (2015). "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations 2015*, pp. 1–15.

Kohavi, R. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection." In: *International Joint Conference on Articial Intelligence (IJCAI)*, pp. 1–7.

Kohavi, R. and G. H. John (1997). "Wrappers for feature subset selection." In: *Artificial Intelligence* 97.1-2, pp. 273–324.

Kumar, V. (2005). "Parallel and distributed computing for cybersecurity." In: *IEEE Distributed Systems Online* 6.10, pp. 1–9.

Längkvist, M., L. Karlsson, and A. Loutfi (2014). "A review of unsupervised feature learning and deep learning for time-series modeling." In: *Pattern Recognition Letters* 42.1, pp. 11–24.

Liddle, J. (2008). *Amazon found every 100ms of latency cost them 1% in sales*. URL: `https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/` (visited on 04/13/2018).

Linden, G. (2006). *Marissa Mayer at Web 2.0*. URL: `http://glinden.blogspot.de/2006/11/marissa-mayer-at-web-20.html` (visited on 03/29/2018).

Liu, C., S. C. H. Hoi, P. Zhao, and J. Sun (2016). "Online arima algorithms for time series prediction." In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1867–1873.

Lorido-Botran, T., J. Miguel-Alonso, and J. A. Lozano (2014). "A review of auto-scaling techniques for elastic applications in cloud environments." In: *Journal of Grid Computing* 12.4, pp. 559–592.

MacCormack, A., R. Lagerstrom, M. Mocker, and C. Y. Baldwin (2017). "Digital agility : The impact of software portfolio architecture on IT system evolution digital agility."

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity." In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133.

McInnes, L., J. Healy, and S. Astels (2017). "hdbscan: Hierarchical density based clustering." In: *Journal of Open Source Software* 2.11, p. 205.

Micchelli, C. A. and M. Pontil (2005). "On learning vector-valued functions." In: *Neural Computation* 17.1, pp. 177–204.

Montgomery, D. (2009). *Introduction to statistical quality control*, p. 754.

Msiza, I. S., F. V. Nelwamondo, and T. Marwala (2007). "Artificial neural networks and support vector machines for water demand time series forecasting." In: *2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 638–643.

Ohta, S (2016). "Obtaining the knowledge of a server performance from non-intrusively measurable metrics." In: *International Journal of Engineering and Technology Innovation* 6.2, pp. 135–151.

Palit, A. K. and D. Popovic (2005). *Computational intelligence in time series forecasting: Theory and engineering applications*. Springer- Verlag New York, Inc., p. 372.

Panwar, M. (2013). "Application performance management emerging trends." In: *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*, pp. 178–182.

Patcha, A. and J. M. Park (2007). "An overview of anomaly detection techniques: Existing solutions and latest technological trends." In: *Computer Networks* 51.12, pp. 3448–3470.

Patel, P., A. Ranabahu, and A. Sheth (2009). *Service level agreement in cloud computing*. Tech. rep.

Peffers, K., T. Tuunanen, M. A. Rothenberger, and S. Chatterjee (2007). "A design science research methodology for information systems research." In: *Journal of Management Information Systems* 24.3, pp. 45–77.

Peng, J, Z Huang, and J Cheng (2017). "A deep recurrent network for web server performance prediction." In: *Proceedings - 2017 IEEE 2nd International Conference on Data Science in Cyberspace, DSC 2017*, pp. 500–504.

Powers, D. M. W. (2011). "Evaluation: From precision, recall and F-measure to roc, informedness, markedness & correlation." In: *Journal of Machine Learning Technologies* 2.1, pp. 37–63.

Rabl, T., M. Sadoghi, H.-A. Jacobsen, S. Gomez-Villamor, V. Muntes-Mulero, and S. Mankovskii (2012). "Solving big data challenges for enterprise application performance management." In: *Proceedings of the 38th Conference on Very Large Databases (VLDB '12)*, pp. 1724–1735.

Raschka, S. (2017). *Python machine learning*. 2nd. Packt Publishing.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, pp. 386–408.

– (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323.6088, pp. 533–536.

Sander, J, M Ester, H. P. P. Kriegel, and X. Xu (1998). "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications." In: *Data Mining and Knowledge Discovery* 194.1, pp. 169–194.

Schubert, E., J. Sander, M. Ester, H. P. Kriegel, and X. Xu (2017). "DBSCAN revisited, revisited." In: *ACM Transactions on Database Systems* 42.3, pp. 1–21.

Shcherbakov, M. V., A. Brebels, N. L. Shcherbakova, A. P. Tyukov, T. A. Janovsky, and V. A. evich Kamaev (2013). "A survey of forecast error measures." In: *World Applied Sciences Journal* 24.24, pp. 171–176.

Shearer, C., H. J. Watson, D. G. Grecich, L. Moss, S. Adelman, K. Hammer, and S. a. Herdlein (2000). "The CRISP-DM model: The new blueprint for data mining." In: *Journal of Data Warehousing* 5.4, pp. 13–22.

Simonyan, K. and A. Zisserman (2015). "Very deep convolutional networks for large-scale image recognition." In: *International Conference on Learning Representations (ICRL)*, pp. 1–14.

Sorjamaa, A. and A. Lendasse (2006). "Time series prediction using DirRec strategy." In: *European Symposium on Artificial Neural Networks*, pp. 143–148.

Spence, C., L. Parra, and P. Sajda (2001). "Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model." In: *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA 2001)*, pp. 3–10.

Stelzer, D. (2010). "Enterprise architecture principles: Literature review and research directions." In: *Service-Oriented Computing ICSOC/ServiceWave 2009 Workshops*. Vol. 6275, pp. 12–21.

Taieb, S. B., G. Bontempi, A. Sorjamaa, and A. Lendasse (2009). "Long-term prediction of time series by combining direct and MIMO strategies." In: *Proceedings of the International Joint Conference on Neural Networks*, pp. 3054–3061.

Takashi, S., K. Eiji, Y. Suguru, and Y. Heiichi (2006). "Performance anomalies of advanced web server architectures in realistic environments." In: *8th International Conference Advanced Communication Technology, ICACT 2006 - Proceedings*. Vol. 1, pp. 169–174.

The Open Group (2009). *TOGAF (The Open Group Architecture Framework) version 9*, p. 778.

Tunis, J. (2016). *3 key metrics for successful web application performance*. URL: `http://www.apmdigest.com/key-metrics-for-successful-web-application-performance-2` (visited on 03/29/2018).

Van Der Raadt, B. and H. Van Vliet (2008). "Designing the enterprise architecture function." In: *4th International Conference on the Quality of Software Architectures (QoSA2008)*. Vol. 5281 LNCS, pp. 103–118.

Wang, J, Y Yan, and J Guo (2016). "Research on the prediction model of CPU utilization based on ARIMA-BP neural network." In: *MATEC Web of Conferences*. Vol. 65.

Watson, M. (2017). *8 key application performance metrics and how to measure them*. URL: `https://stackify.com/application-performance-metrics/` (visited on 03/29/2018).

Webster, J. and R. T. Watson (2002). "Analyzing the past to prepare for the future: Writing a literature review." In: *MIS Quarterly* 26.2, pp. xiii –xxiii.

Werbos, P. J. (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis.

Wert, A., J. Happe, and L. Happe (2013). "Supporting swift reaction: Automatically uncovering performance problems by systematic experiments." In: *Proceedings - International Conference on Software Engineering*, pp. 552–561.

Xu, W.-B., Y Qi, and D Hou (2010). "Multi-dimensional time series-based application server aging model." In: *2010 International Conference on Wavelet Analysis and Pattern Recognition, ICWAPR 2010*, pp. 238–243.

Yan, H., Y. Jiang, J. Zheng, C. Peng, and Q. Li (2006). "A multilayer perceptron-based medical decision support system for heart disease diagnosis." In: *Expert Systems with Applications* 30.2, pp. 272–281.

Yanggratoke, R, G Kreitz, M Goldmann, and R Stadler (2012). "Predicting response times for the Spotify backend." In: *8th International Conference on Network and Service Management (CNSM 2012)*, pp. 117–125.

Zhang, X, C Feng, and G Wang (2014). "Prediction of website response time based on support vector machine." In: *Proceedings - 2014 7th International Congress on Image and Signal Processing, CISP 2014*, pp. 912–917.