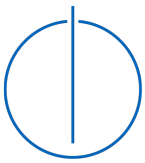
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Comparison of Anomaly Detection Frameworks for Time-Series data from Car Monitoring

Marko Stapfner



DEPARTMENT OF INFORMATICS

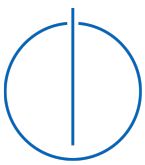
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Comparison of Anomaly Detection Frameworks for Time-Series data from Car Monitoring

Vergleich von Anomaly Detection Frameworks für Zeitreihen von Betriebsdaten von Automobilen

Author:	Marko Stapfner
Supervisor:	Prof. Dr. Michael Gerndt
Advisor:	Anshul Jindal
Submission Date:	15th February 2022



Acknowledgments

I want to thank my advisor Anshul Jindal for his support during my thesis, providing me always help, and for having an open ear for my questions. His feedback was very valuable and helped me improve my thesis. I would also like to thank the monitoring team from the car manufacturer, who helped me to query and review the existing data.

Also, I express my gratitude to Prof. Dr. Michael Gerndt for his seminar Cloud Computing which increased my interest in the topic of monitoring cloud infrastructures and helped me in the early phase of finding and scoping my topic.

Abstract

The software industry experienced tremendous growth, together with the complexity of the used software systems in professional enterprises. More and more car manufacturers move their software systems to the cloud. This results in more demand for professional monitoring solutions. The market provides different monitoring solutions dedicated to special needs, and Prometheus is a well-known and widespread time-series monitoring solution. Time-series monitoring often only provides a view of past or current events and metrics, but more intelligent solutions are required to fulfill the challenge of ever-growing complexity.

This thesis focuses on building a flexible anomaly detection integration platform (called FADIP) that allows the evaluation and usage of different anomaly detection algorithms on various timeseries from car monitoring (unlabeled and labeled) and performing this evaluation. Functional and non-functional requirements for the platform are presented, as well as a fair comparison methodology. It is shown that the MCC score and specificity of the algorithms are the relevant metrics to compare algorithms for productive operation, and the algorithms iForest, HBOS, KNN, CBLOF, and COPOD are compared on selected datasets. The evaluation results show that the time-series differ significantly, so every algorithm needs to be assessed before using it in the infrastructure.

Contents

1 Introduction

1.1 Introduction to Monitoring

Over the last years, the software industry experienced tremendous growth in various aspects [30]. The demand for professional enterprise software is continuously growing, with an estimate of 669,82 billion us dollars in revenue predicted for 2022 by Statista [26]. Not only the demand for software is growing. The increasing amount of IoT devices [14], the ever-advancing digitisation of a wide variety of industries leads to a complexity that is not manageable by humans. Especially in the last years, cloud computing has became a real and relevant trend, helping corporates move their applications into the offered target environments (Infrastructure-as-a-service, Platform-as-a-service, Software-as-a-service, Functions-as-a-service).

In modern computational cloud infrastructure, the demand for professional monitoring solutions continuously grows, together with the number of servers, load-balancers, and other infrastructure. Applications also tend to be more geographically distributed. Different challenges come with these demands, for example, fast processing, storage, and visualization of the monitoring data. Also, user-interface, provided functionality, alerting, and help-desk integration play an important role. [13]

Monitoring can be precisely defined as the "tools and processes by which you measure and manage your technology systems" [34]. These technology systems often involve many different applications, services, systems, tools, processes, and connections. This heterogeneity poses different requirements to the monitoring tool (in terms of flexibility & stability). According to James Turnbull, Monitoring has two customers, the technology and the business. [34] The technology stands for the operation team that maintains the technological system, often called DevOps or Site Reliability engineering. Monitoring supports the fault detection, resolution, and other potential problems by giving information about the current or past state of the systems and measuring performance metrics to ensure the operational quality, in the best case before there is an impact on the internal customer (business). [34] The business customer represents the business team and their processes that should be supported. They gain value from having detailed metrics of the systems used in the business processes and can help plan investments into product & technology, as well as demonstrate the value proposition of the monitored systems. [34]

There are different monitoring tools and software available on the market dedicated to special needs. General-purpose paid monitoring tools like DataDog, NewRelic One, Nagios XI, Dynatrace are known as well as specialized tools like InfluxDB as a time-series database, SCOM for Microsoft-based infrastructures, or PRTG Network Monitor for infrastructure and networks. [16]

Additionally to these solutions, open-source software solutions and open-source variants of enterprise products are available. Especially in cloud & microservices infrastructures, one very known and used monitoring solution is Prometheus. Prometheus was developed in 2012 by Matt Proud and Julius Volz at Soundcloud using the Go programming language, and in 2015 it was open-sourced on GitHub. Prometheus is architected as a system that receives the monitoring data of the systems via so-called exporters that prepare the data so that Prometheus can process and store it. [38]

Prometheus itself stores the data in a self-developed high efficient format on the local disk. The basic format is that Prometheus stores time-series data, so every datapoint consists of a millisecond-precision timestamp together with the metric value (float64 data type) and some labels. Every time-series has a unique metric name (identifier) and optional key-value pairs (labels, key, and value are of data type string). [6]

1.2 Challenges of Monitoring

Monitoring involves a variety of challenges and goals. Monitoring does not only involve monitoring the application software but also network infrastructure, hardware, or virtual machines. There are also discussions on how to monitor the monitoring system, with different approaches, for example, multiple monitoring stacks that monitor each other or rely on continuous alerting.

Under the assumption that the monitoring system can handle the load of collecting metrics from many systems, long-term storage (more than a few months) is a challenge. The monitoring tool used in this thesis, Prometheus, is not specialized in long-term storage. It recommends only keeping the data for about 60 days in the internal time-series database. It can be configured to hold it for a longer time period, but the performance when querying the data can drop significantly. This is a notable challenge because, as shown in the section before, the system complexity is increasing, increasing the complexity of monitoring simultaneously. There are other options for storing the time-series data longer, for example Cortexmetrics, Thanos.io or victoriametrics [27]. In the monitoring stack used in this thesis, Thanos.io was the choice. Thanos is a flexible, microservice-based approach that solves the problem of long-term storage. It is deployed alongside the existing Prometheus setup, and there are no changes required to the existing monitoring stack [31]. Thanos.io uses different microservices for writing,

accessing & caching the data and allows to use object storages like AWS S3. There are no upper bounds in terms of storage; AWS S3 can be continuously used, and scaling or reconfiguring is not necessary (and not even possible) [36]. The storage challenge is solved sufficiently by adding Thanos.io to the Prometheus stack. [32]

From the challenge of storing monitoring data, accessing the data and visualizing it in an appropriate time window can also be problematic. A standard configuration of Prometheus fulfils these requirements, but Prometheus can only access the stored metrics from its own TSDB. Thanos provides a remedy here as well, allowing data from different Prometheus instances and object stores to be queried through the query microservice [33].

An important problem of monitoring is that in most cases, it only provides a view on the past or current events, parameters, and metrics. This leads to the problem that software reliability engineers need to manually check for trends and (potentially) upcoming problems by watching the relevant metrics and analyzing their trends. This is not always possible, since in increasingly complex software infrastructures, it is tough to always have an overview of the interconnections and dependencies between all components and their impact on the overall performance of the software systems (performance in the context of business performance, not application performance). As a result, it is challenging to have a continuous outlook into the future and to identify trends and potential problems or vulnerabilities.

A potential solution for the problem just outlined is anomaly detection (or also called: outlier detection). It helps the software reliability engineers and DevOps engineers to automate certain parts of monitoring and supports a pro-active approach on the monitoring and early detection of faulty behaviour of components and systems.

1.3 Research Objectives and Main Contributions

The main research objective of this thesis is to develop a flexible anomaly detection integration platform that allows to evaluate & use different available anomaly detection algorithms on open-source and monitoring datasets from a known German car manufacturer. The design of the platform is geared towards use in the monitoring infrastructure of a car manufacturers backend.

- **RO-1:** Define objective and fair comparison criteria for available anomaly detection algorithms.
- **RO-2:** Formulate functional and non-functional requirements that the anomaly detection platform needs to fulfill for productive usage in professional cloud infrastructure.

- **RO-3:** Provide an evaluation of different anomaly detection algorithms and their suitability for productive usage in professional cloud infrastructure.

The key contributions of this thesis include:

- We develop a flexible anomaly detection integration platform (called FADIP) that allows to evaluate and use different anomaly detection algorithms in the application domain of monitoring complex backend infrastructure. We present the integration into the current monitoring stack, the configuration process, and explain the detailed functionality. Non-functional and functional requirements to the platform are defined, and the compliance to them is shown.
- We present a fair methodology and comparison criteria to compare different unsupervised anomaly-detection algorithms for univariate time-series of the monitoring data from car manufacturers.
- Lastly, a short overview of known anomaly detection algorithms and their way of function is provided, together with an analysis and evaluation of the algorithms on different open-source datasets and monitoring data from a car manufacturer's backend.

1.4 Outline of the Thesis

This thesis is structured as follows:

- Chapter 2 provides some background information on the topic anomaly detection, as well as an analysis of the application. The application domain itself is divided into an explanation of the application delivery controllers that are monitored and the stack that collects the monitoring data.
- Chapter 3 presents related work in the anomaly detection area. First, an overview is given about basic machine learning concepts, then, the used anomaly detection algorithms are explained based on their functionality.
- Chapter 4 explains the methodology used to evaluate the anomaly detection algorithms, as well as the defined functional and non-functional criteria.
- Chapter 5 provides information about how the platform is structured and developed, the overall architecture, and the configuration and integration options. This chapter also defines requirements that the platform fulfills for a professional usage in distributed cloud environments.

- Chapter 6 presents the results of the evaluations on different datasets. First, the used datasets are explained, and then the results of the anomaly detection algorithms on these datasets is explained. This chapter also provides an assessment of which algorithms are suitable for professional usage in the specified application. Also, different improvements on anomaly detection are described.
- Chapter 7 summarizes and concludes the thesis with an outlook on the potential options and improvements.

2 Background

2.1 Anomaly Detection

There exist different definitions of anomalies, each tailored to the right scope. In this thesis, we rely on the definition provided by Lavin and Ahmad: *"We define anomalies in a data stream to be patterns that do not conform to past patterns of behavior for the stream. This definition encompasses both point anomalies (or spatial anomalies) as well as temporal anomalies."* [20]. This definition focuses on detecting of anomalies for the past, not including other data input types that could identify anomalies, e.g., past outliers or unwanted system behaviour.

For a better definition, we first adapt the definition from Shiblee Sadik and Le Gruenwald to time-series: A time-series *"is an infinite set of data points, $P = \{D_t | 0 \leq t\}$ where a data point D_t is a set of attribute values with an explicit or implicit timestamp. Formally a data point is $D_t = (V, \tau_t)$ where V is a p-tuple, each value of which corresponds to one attribute, and τ_t is the timestamp for the t-th data point."* [28].

Therefore, this thesis extends the definition of anomalies and sets the focus to time-series data: We define: *Anomalies in time-series data are a collection of data points or a single data point that shows a deviation of the monitored metric from the expected behaviour*. This definition seems vaguer, but its advantage is that it also considers trends independent of seasonal behaviour and sets the focus to the unexpected behaviour on the underlying monitored system. Of course, this definition is only useful, when it is guaranteed that the behaviour and state of the monitored system are present in the available monitoring data. Another advantage of this definition is that it also captures the problem of concept drift described by Nan Jiang and Le Gruenwald in their paper [17], since the focus of the definition of anomalies is on the system behind it. Concept drift means that the distribution of data in a data stream will change over time (and this is not necessarily related to an anomaly), and of course, all anomaly detection algorithms should be able to tolerate this behaviour and continue to detect anomalies accurately and successfully.

2.1.1 Anomaly types

There exist different types of anomalies (as they appear in the datasets), and they can be categorized into three main categories [28]:

- *Individual Outliers in the context of the complete dataset*. This type of outlier represents an individual outlier in the whole dataset context and extreme derivation, for example, an incredible peak in a dataset that describes the number of requests to a web-server of an online-shop. Identifying this outlier is compared to the other types relatively easy since the outlier deviates strongly from all other points.
- *Individual Outliers in sub-context*. This type is also a single datapoint like the first type, but it deviates from the typical behaviour of similar datapoints instead of the whole dataset. An example would be the number of requests to a web server that, for some reason, is twice as high on a Tuesday afternoon as on all other Tuesdays in the afternoon.
- *Grouped Outliers*. Grouped outliers are a group of datapoints that form an outlier together, and they differ in that the individual data points are not anomalies when viewed alone in the context of the entire dataset, but their arrangement as a group is.

More precisely, anomaly detection is a classification problem; an algorithm has to divide a single or a series of points into two classes: Normal points and anomaly points. The context or scope (or application domain) is relevant like all classification problems.

2.2 Application Domain

As described in Chapter 1, the application domain is derived from the increasing complexity of technical infrastructure. The automotive industry is also benefiting from increasing digitisation, and in recent years the vehicles produced have become more and more digital. Digital car experiences are not possible without cloud infrastructure operated by car companies or their suppliers. [18]. As Holger Breuing and his colleagues show in their report [4], car manufacturers face lots of different challenges, and the cloud infrastructure for different digital services (like car provisioning, digital entertainment services, navigation, must guarantee strong safety and security. Digital infotainment services in particular, such as navigation, music, communication, and concierge, require related services in the backend of the car manufacturer that provide all the data for the functionality. Many different vehicle generations, models, model variants, and different equipment make for a correspondingly complicated infrastructure (assuming that all vehicles are provided with updates and services even years after delivery).

This master thesis focuses on the monitoring data of the application delivery controllers and load-balancers, which perform various tasks, but mainly the routing of vehicle requests to the corresponding systems.

2.2.1 Application Delivery Controller

The corresponding infrastructure that is the base of the monitoring data & systems is shown in 2.1 in a simplified version. The basic operation can be described with the following example: The infotainment system asks the car manufacturer backend for the current data on traffic jams and accidents for a navigation route in order to adjust the route planning. The application delivery controllers accept this request, and after safety protocols have been executed, it is decided which service will take over the response to this request based on the requested data and the requesting vehicle. This request is then forwarded to the service located in the cloud. After processing the request, the service sends the response back to the application delivery controller, returning it to the vehicle.

All application delivery controllers expose endpoints for a monitoring system to collect metrics. For security reasons, the detailed operating mode and a more detailed analysis of the exposed monitoring data is not possible, but an anonymized description of the dataset is in chapter 6.

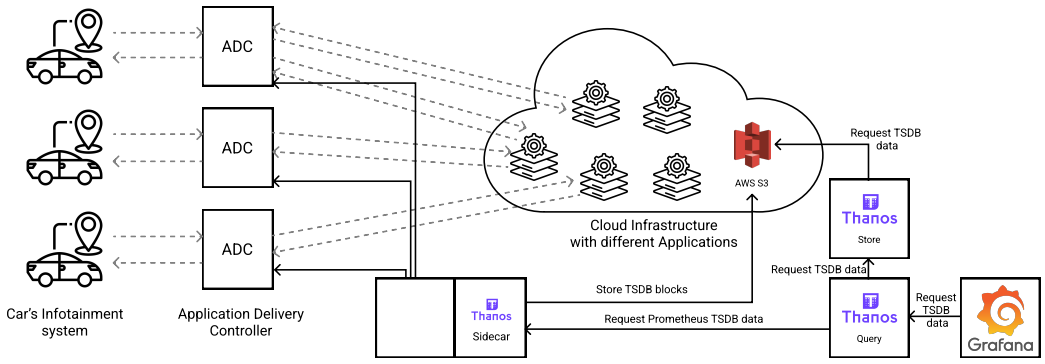


Figure 2.1: Simplified Overview of the infrastructure

2.2.2 Monitoring Stack

The monitoring stack in 2.1 relies mainly on Prometheus as a supplier of time-series data. Prometheus is an open-source monitoring solution that collects metrics of different systems directly or via so-called exporters. Prometheus stores the monitoring data in its own Time-series-Database in the TSDB format in blocks for every two hours. Since Prometheus is not suitable as a long-term storage solution, Thanos takes over this role. Thanos is deployed in a microservice approach, in the base setup only consisting of these components: Thanos Sidecar, Thanos Store, and Thanos Query. Thanos Sidecar is a component deployed on the same instance with the Prometheus instance and allows

other Thanos Components to use the Prometheus-API in the same way as other Thanos Store APIs. Thanos Sidecar also reads the TSDB blocks written to the disk storage (that contain the time-series data) and uploads them to an object storage solution like Amazon Web Services S3 or Google Cloud Storage Buckets. The Thanos Store component is a "wrapper" or interface for object storage bucket to allow querying the data inside the bucket. Thanos Query is the component that exposes all Thanos microservices (and connected Prometheus and storage buckets) as PromQL-compatible Prometheus HTTP v1 API. Thanos Query translates a PromQL request into the internal Thanos StoreAPI, and this protocol is then used to query all components registered in the query component. The registered components (mostly Thanos Sidecar and Thanos Store) then translate the request, collect the data and send it back to Thanos Query. Thanos Query then assembles it and returns it in a Prometheus-compatible format.

3 Related Work

3.1 Machine-Learning Concepts

To better understand the algorithms, the machine learning concepts are first explained in this section. Mehryar Mohri and colleagues define machine learning as *"Machine learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions."* in their book [25]. They consider the past information provided to the algorithm during learning as experience.

More precisely, past information can be described as data, divided into two different forms. A basic distinction is made between *structured* and *unstructured data*. Structured data means that the data is represented in a structured way, like in rows and columns in a spreadsheet or a database table. A single datum (a row) consists of different features or attributes (a value inside a column for that row). Unstructured data are not in such a format or have very different dimensions. Examples of unstructured data are images or text documents. Machine Learning can operate on both of these data types (depending on the algorithms), but in general, machine learning algorithms perform better on structured data. [29]

The subsequent differentiation concerns the machine learning algorithms themselves, where a distinction can be made between unsupervised and supervised learning.

- *Supervised learning* means that the algorithm is fed with data that contains exactly the target attribute that is being searched for in the unknown data. In this way, the algorithm is given help in making a decision or predicting characteristics for new data instances. [29]

  Supervised learning is used in two different forms, classification and regression. Classification means to predict a discrete state, value, or category; an example is dividing data points into anomalies and normal data points. This example is called binary classification, but there is also the distinction of multiple categories or that a single datum can be classified into multiple categories (this is called multi-label classification). On the other hand, regression means to predict a numerical value, so especially in the best case, the algorithm tries to find a formula with which a good estimate of the true value can be found. [29]

- *Semi-supervised learning* in the context of anomaly detection means that the machine learning algorithm is trained using a dataset that contains only training data that belong to a single class, in this case, the class normal points. Thus, the training data contains only normal datapoints and no anomalies, so the machine learning algorithm can build a model of the normal data and detect anomalies based on derivation from this model. [11]
- In *unsupervised learning*, it is not known in advance what the correct answer is for the training dataset, so the machine learning algorithm itself has to find interesting correlations. A good example of unsupervised learning is clustering, where data is divided into several groups based on certain attributes. [29]

The main advantage of unsupervised machine learning algorithms is that they do not need labeled data as input. The use case clearly shows that all the input data collected by the monitoring stack does not include labels. In this thesis, the focus is on unsupervised learning as a classification problem with two categories, *anomalous* and *normal*. *Anomalies* and *outliers* are used as synonyms, on an implementation level, 0 denotes *normal* and 1 denotes *anomalous*.

3.2 Anomaly Detection Algorithms

A tremendous amount of anomaly (or outlier detection) methods, algorithms, and frameworks exist. These algorithms rely on a wide variety of mathematical concepts and methods, and their operation is sometimes fundamentally different. Some algorithms work better for particular purposes; others are more useful for general applications. For this reason, a selection of known anomaly detection algorithms is explained in outline in this section so that the evaluations obtained in chapter 6 can be put into context.

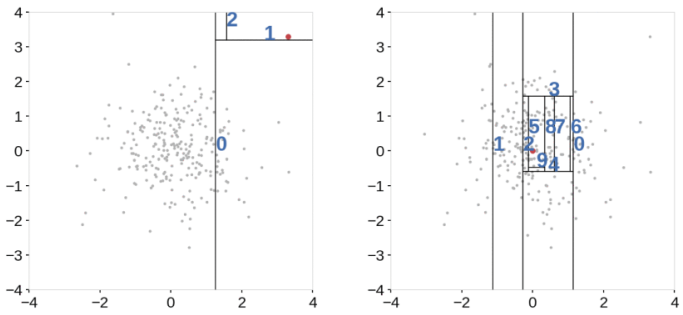
3.2.1 iForest - Isolation Forest

Isolation Forest (iForest) is an unsupervised anomaly detection algorithm that works fundamentally different than most of the mentioned anomaly detection algorithms. As every anomaly detection algorithm, it needs to detect anomalous instances as fast as possible with high precision and minimal space consumption. As the this section shows, most of them rely on distance- or density-based measures to identify anomalous points or patterns in data. Also, most algorithms focus on building a classifier for normal (non-anomalous) instances and then distinguish the anomaly points from the normal points.

Isolation Forest takes a different approach. It relies on the concept of isolation, meaning that anomalies are few and different, and therefore they are susceptible to this isolation. [22]

This leads to the fact that anomaly detection in iForest is not a "byproduct"; it is the sole purpose. Liu, Fei Tony and colleagues state in their paper, iForest is more accurate in the detection of anomalies (by means of false-positives and false-negatives) and can exploit subsampling. iForest is not using any distance or density measures, so the computational cost of the isolation-based anomaly detection is lower. Also, iForest operates with linear time complexity while being able to handle enormous data sizes and high-dimensional problems.[22]

Isolation Forest works by separating points from the rest of the other points. iForest utilizes binary trees with a recursive instance partition to achieve this isolation. The advantage is that the binary trees produce shorter paths for anomalous points because less anomalies are in a partition. Also, anomaly instances are more likely to be separated earlier than the normal instances in the process. To gain more reliability, not only a single binary search tree is used, a "forest" of trees is used and the average path length of the points is used to determine the anomalies. As Liu, Fei Tony and colleagues state in their paper, a normal point requires more partitions to be separated than an anomalous point. It can also be seen in figure 3.1.



(a) Anomaly point (b) Nominal point

Figure 3.1: Isolating an anomalous point and a normal point, image from [15]

Compared to distance- and density-based anomaly detection algorithms, iForest can also detect anomaly classes where traditional algorithms struggle. When relying on density-based anomaly detection, the "ground-truth" is that anomalous points occur in dense regions and anomalies in sparse regions. With distance-based anomaly detection, the ground truth is that normal points are closer to their neighbors and anomaly points are further away. In reality, these two assumptions should always apply to all points, but there can be exceptions; for example, points with low density and long distance are

not necessarily anomalies. [22]

Problems with the detection can also happen when these two metrics are measured locally, but anomalies appear globally, especially when the dataset includes regions with different densities. iForest should perform better with these datasets since the path length grows in an adaptive context, advancing from global to local context, while density and distance only concern the local context. This leads to the fact that isolating anomalies allows detecting clustered and scattered anomalies, and distance and density can only detect scattered anomalies. [22]

3.2.2 KNN - K-Nearest-Neighbor

K-Nearest-Neighbor-based outlier detection is a distance-based outlier detection algorithm that uses a metric called "weight" that represents the sum of the distances from the k nearest neighbors of a point. The algorithm is unsupervised, so the data used for learning and prediction can be unlabeled. This algorithm identifies outliers as the points with the largest "weight" values. [2]

To compute the distances to the k nearest neighbors, first, the k nearest neighbors need to be identified. This is done by fitting the d-dimensional dataset into a hypercube $D = [0,1]^d$ and mapping it to the interval $I = [0,1]$ using the Hilbert space-filling curve and identifying the neighbors by taking the predecessors and successors of a point on I. The Hilbert space-filling curve has the property that if two points are close in I, they are also in D, but the other way round is not always true. Due to this fact, there can be a loss in the points near each other. Therefore, the dataset is shied $d+1$ times along the main diagonal of a hypercube $[0,2]^d$. In the figure 3.2, the iterations of the Hilbert space-filling curve can be seen. Then, the distance is computed in the multi-dimensional D space, and only the neighbors are identified using the one-dimensional space. [2]

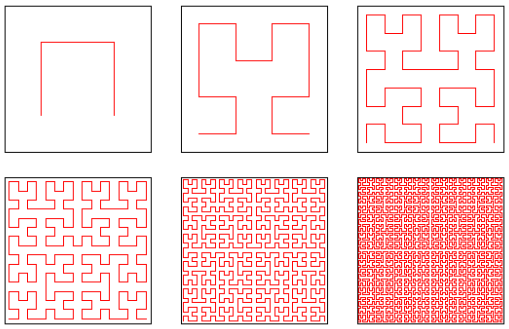


Figure 3.2: First 6 iterations of the Hilbert Space Filling Curve

In detail, the algorithm consists of two phases. In the first phase, it uses a fast

approximation to find a solution and iterates several at most $d + 1$ times through the dataset. In every step, a better lower bound for the weight of potential outliers is calculated. With every iteration, the number of anomaly candidates decreases. In the second phase, the algorithm calculates the exact solution in a single scan of only a small fraction of the dataset. In their paper, F. Anguilli and C.Pizzuti showed that after $\bar{d}$ steps (with $\bar{d} << d + 1$), the algorithm finds the exact solution. [2]

3.2.3 HBOS - Histogram-based Outlier Scores

Histogram-based Outlier Scores (HBOS) is an unsupervised anomaly detection algorithm based on histograms. Like iForest, it considers the fact that anomalies are rare and different (in terms of features). Due to its nature, HBOS detects anomalies very fast in large datasets compared to supervised or semi-supervised anomaly detection algorithms. [10]

In detail, HBOS calculates for every datapoint a histogram-based outlier score. It can handle multivariate data but does not take into account possible correlations between the features. The first step is the calculation of univariate histograms for every single feature (isolated from other features). The histograms are generated depending on the type of feature; for categorical data, a simple value count of the appearances of data points in the category is used. For numerical data, two options can be used for histogram creation. [10]

Every histogram contains some "bins" or containers, and every container represents a single category or an interval of numerical values. The number of instances is counted for every container, and all "bins" of a dataset together form a histogram. Histograms can be created with constant bin-width (the interval width stays the same for all bins), or the interval width adjusts on a criterion.

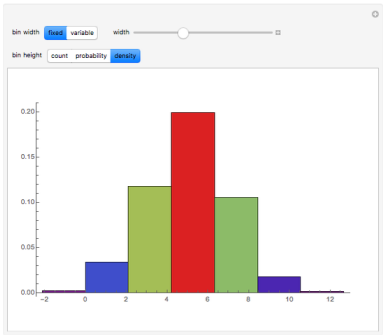


Figure 3.3: Example histogram with static bin length, image from [7]

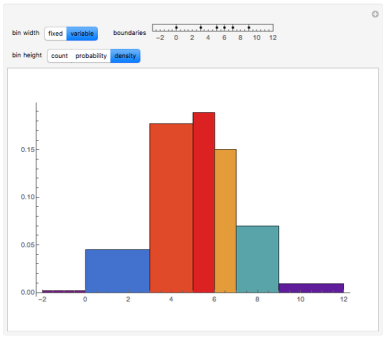


Figure 3.4: Example histogram with dynamic bin length, image from [7]

HBOS can use both options, fixed bin-width histograms and dynamic bin-width histograms. For the fixed bin-width histograms, k equally sized bins are created for the complete value interval of the data, and the count of points in every bin (height of the bin) represents an estimation of the density. The dynamic bin-width histograms work by sorting the values of all datapoints and then grouping them based on: $\frac{N}{k}$ with N as the number of total data points and k the number of bins. This means, for static bins, the size (value interval) of the bins is fixed, and for the dynamic intervals, the amount of datapoints in each bin is fixed and the interval from the lowest value of all datapoints inside this bin until the highest value of all datapoints inside this bin. [10]

All computed feature-histograms are normalized so that the maximum hight is 1.0 to weight the single features equally for the outlier score. The exact outlier score is then calculated by using this formula: $HBOS(p) = \sum_{i=0}^{d} log(\frac{1}{hist_i(p)})$ with d the amount of features, p a datapoint and $hist_i(p)$ the height of the bin of the datapoint p. [10]

As Markus Goldsetin and Andreas Dengel state in their paper, in comparison to other anomaly detection algorithms, HBOS performs well on global anomalies and fails on local anomalies (because modeling local outliers with density estimation is not possible with histograms). HBOS did the outlier calculation significantly faster than some nearest-neighbor-based and k-means based algorithms. The time complexity for HBOS is linear $O(n)$ for static bin width and $O(n * log(n))$ for dynamic bin width. [10]

3.2.4 CBLOF - Cluster-based Local Outlier Factor

Cluster-based Local Outlier Factor is a proximity-based anomaly detection algorithm that differs from the typical outlier detection algorithms as well as from the typical clustering-based methods. First of all, CBLOF is based on a different definition of an outlier. The algorithm identifies large and small clusters (by using an external clustering algorithm) and then calculates the cluster-based local outlier factor by measuring the

size of the cluster the datapoint belongs to together with the distance between the datapoint and the nearest cluster (if datapoint lies in a small cluster) [12]

In Detail, the algorithm consists of two phases. In the first step, the algorithm tries to identify the existing clusters. Therefore, the authors Z. He, X. Xu, and S. Deng suggest using an existing clustering algorithm, the squeezer algorithm.

In the next step, the identified clusters are categorized by two assumptions into small and large clusters. The first assumption is that most data points in the dataset are not outliers, so clusters that hold a big amount of datapoints can be identified as clusters. Secondly, large and small clusters have a significant size difference, so small clusters can be identified by comparing to other clusters (and of course the reverse way is also valid). [12]

For separating the clusters into large and small clusters, it is necessary that CBLOF takes as input two parameters α and β in two formulas. For the following formulas, $C = \{C_1, C_2, ..., C_k\}$ is a set of clusters that is sorted like: $|C_1| \geq |C_2| \geq ... \geq |C_k|$.

$$(|C_1| + |C_2| + ... + |C_b|) \geq |D|^* \alpha$$

The parameter α decides how many percent of all datapoints need to be in a cluster to have a cluster seen as a large cluster. Some clusters can meet this assumption, and for categorizing the remaining clusters, this formula is used:

$$\frac{|C_b|}{|C_{b+1}|} \geq \beta$$

The parameter β is a relationship multiplier; this means that a cluster is a small cluster if it is β-times smaller than a large cluster. [12]

The calculation of the cluster-based outlier factor of a datapoint depends on the distance between the record and the closest cluster (if datapoint is part of small cluster):

$$CBLOF(t) = |C_i|^* min(distance(t, C_j))$$

where

$$t \in C_i, C_i \in SC$$

and

$$C_j \in LC$$

for

$$j = 1$$

to

$$b$$

and if the datapoint is part of a large cluster, the distance between the record and the own cluster:

$CBLOF(t) = |C_i|^*(distance(t, C_i))$ where $t \in C_i$ and $C_i \in LC$

3.2.5 COPOD - Copula-based Outlier Detection

Copula-based outlier detection is a very new algorithm presented in 2020. The algorithm from Zheng Li and colleagues is based on the concept of copulas used to model multivariate data distribution, and according to their evaluations, the algorithm outperforms most of the other outlier-detection algorithms on different benchmark datasets. [21]

The outlier detection with copulas consists of three steps. In the first step, empirical cumulative distribution functions (ECDFs) are constructed using the training dataset. Next, the ECDFs are used to produce an empirical copula function. The empirical copula function is then used to approximate the tail probabilities for every new datapoint. [21]

"Copulas are functions that enable us to separate marginal distributions from the dependency structure of a given multivariate distribution." [21]. Uniform distributions can be transformed into any desired distributions via inverse sampling, which leads to the fact that a copula allows the description of a joint distribution of $(X_1, ..., X_d)$ using only their marginals. Sklar's Theorem guarantees the existence "of a copula for any given multivariate CDF with continuous marginals." [21]

To let COPOD perform the outlier detection without parameters, an approach using the fitting of empirical cumulative distribution functions (ECDFs) is used (called Empirical Copula). This result of this approach is then used to produce an empirical copula function. This empirical copulation function is then used to calculate the tail probability for each x_i:

$$F_X(x_i) = \mathbb{P}(X_1 \leq x_(1, i), ..., X_d \leq x_(d, i)$$

COPOD relies on extremity comparison to identify outliers, therefore outliers are tail events, so for every datapoint x_i, the probability of having another datapoint that is at least as extreme as x_i. If x_i is an outlier, the probability of observing a similarly extreme value as x_i should be low, and if x_i is not an outlier, the probability should be high. x_i is distributed using to a d-variate distribution function F_X, and the probabilities to observe another similar point are $F_X(x_i) = \mathbb{P}(X \leq x_i)$ and $1 - F_X(x_i) = \mathbb{P}(X \geq x_i)$. $F_X(x_i)$ is defined as the *left tail probability* and $1 - F_X(x_i)$ is defined as the *right tail probability*. [21]

Both tail probabilities need skewness correction to correct the outcome when the outliers can only be found on one extreme end of the data distribution. Zheng Li and colleagues illustrate this in figure 3.5.

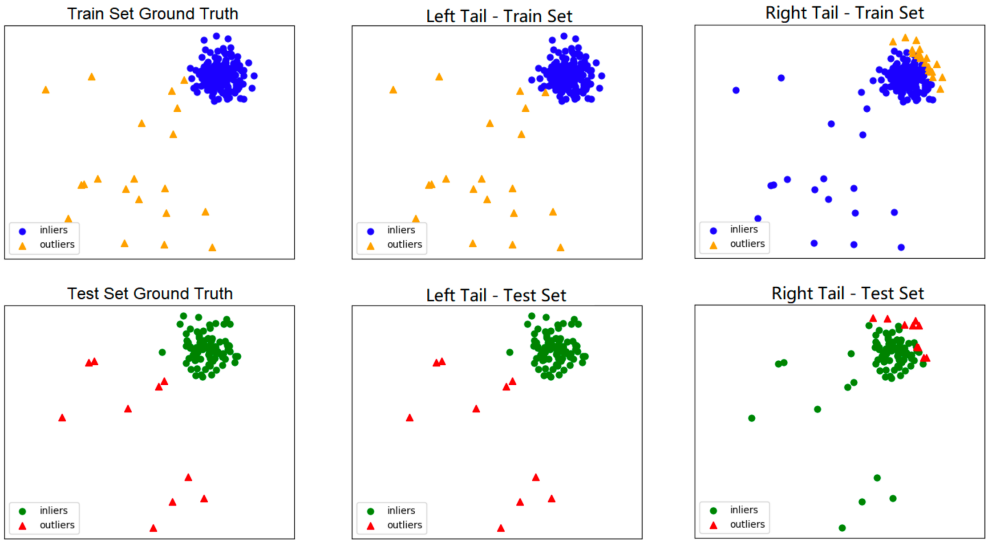


Figure 3.5: Different tail probabilities of example dataset, image from [21]

Figure 3.5 shows a dataset with the ground truth being distributed over a 2-dimensional space, the outliers in the left section of the space, and a cluster with non-anomalous datapoints on the top right. The images in the middle show that using only the left tail probabilities of the COPOD, the detection of outliers was almost 100% precise. The images on the right show that using only the right tail probabilities of COPOD, the identification of outliers was utterly misleading. This demonstrates the need for skewness correction. [21]

Averaging the left and right tail probabilities does not provide an appropriate solution. It just identifies half of the left outliers and half of the right outliers. Averaging is only a viable solution for large and small outliers in the dataset. Since the skewness of the dataset plays a significant role in weighting the left or right tail ECDFs, and depending on the skew of the marginals, the left or right tail probability should be used. In figure 3.5, the left tail is longer, and the mass of the distribution is on the right, so there, the left tail ECDF should be used. [21]

The algorithm takes as input parameter a d-dimensional dataset and produces an outlier score vector with scores between $(0, \infty)$. The outlier scores are not probabilities but a relative measure compared to the other datapoints, with a higher outlier score meaning a higher indication to be an anomalous point. [21]

COPOD as an anomaly-detection algorithm is able to handle very high dimensional data with lots of datapoints in the dataset ($1,000,000$) reliably in the different aspects: ROC-AUC, precision, computation time. [21]

4 Methodology

For a valid comparison of the selected anomaly detection algorithms and frameworks on the chosen datasets and datasources, it is necessary to define fair and measurable comparison criteria. The criteria are divided into functional and non-functional criteria. Functional criteria measure the direct functional behaviour and performance of the algorithm on the dataset via statistical and mathematical methods and relations. The non-functional criteria focus more on the effort of setting the algorithm up, time until the first results can be seen, options for tuning, and usability.

4.1 Functional Criteria

The goal of defining the functional criteria is to create metrics that allow a fair, good comparison. Since anomaly detection is a classification problem with two classes *normal*, denoted as 0, and *anomalous*, denoted as 1, there needs to be a ground truth for every dataset. So defined comparison criteria can only be applied to labeled datasets (labeled by normal and anomalous datapoints). As shown in [19], in this thesis, the confusion matrix is used, producing the 4 base metrics: *true positives*, *false positives*, *true negatives* and *false negatives*. Figure 4.1 shows these 4 metrics.

		Predicted Outlier	
		True	False
Actual Outlier	True	True positive tp	False Negative fn
	False	False positive fp	True Negative tn

Table 4.1: Confusion Matrix

Based on *true positives*, *false positives*, *true negatives*, *false negatives*, 5 more known metrics are used for comparison of binary classifications [19]. *Recall* or *sensitivity*, which means, out of all datapoints labeled as anomalies, how many got identified as anomalies by the algorithm, can be seen in 4.1. The metric *precision* stands for the amount of true anomalous datapoints of all the datapoints that were predicited as anomalies 4.1. *Specificity*, shown in 4.1, represents the proportion of negatives, which are correctly identified [19].

$$precision = \frac{tp}{tp + fp} \quad (4.1) \qquad recall = \frac{tp}{tp + fn} \quad (4.2) \qquad specificity = \frac{tn}{tn + fp} \quad (4.3)$$

Figure 4.1: Formulas for *precision*, *recall* and $specificity$

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \quad (4.4) \qquad f1 - score = \frac{2}{precision^{-1} + recall^{-1}} \quad (4.5)$$

Figure 4.2: Formulas for *accuracy* and $f1 - score$

From the two metrics just mentioned, "precision" and "recall", the f1-score can be calculated, which represents both in balance. Figure 4.2 shows the exact formula. [9] The same figure 4.2 also shows the metric accuracy, which represents the overall percentage of correct predictions, so how many percent of the classifications were right. However, this metric is only meaningful for symmetric datasets (with an equal amount of normal and anomalous datapoints), for example, with a dataset with 80% normal points and 20% anomalous points and a detector that outputs only normal instances, the *accuracy* returns 80%. However, the *precision* and *recall* would be 0, both the possibly worst values. [35]

On closer inspection, the F1 score shows an interesting problem: it depends very much on the distribution of binary classes in the training and test dataset and also on the contamination rate at which training is done. As Damien Fourure and his colleagues show in their paper [9], there is a threshold needed for a classifier to divide the datapoints into the two classes, and a standard solution for threshold computation is to use the contamination rate of the test- or training-dataset. The theory for comparing machine learning models is to always separate train and test sets and use the labeled training set contamination rate for the training of the anomaly detection algorithm. Looking at unsupervised anomaly detection algorithms, the drawback of this approach is that the anomalous points are only used for the contamination rate calculation, but for the actual training this information is thrown away. The evaluation gets more precise by manually inserting these points into the test set or increasing the number of anomalies in the test set. This results in the fact that the *f1 score* improves with the number of anomalies in the test set, allows a better representation of the algorithm, and reduces the general comparability of algorithms, especially when using different datasets.

This problem is solved by another evaluation metric, Matthews Correlation Coefficient (*MCC*), shown in 4.3. This metric uses the normal points (*true negatives* and anomalous points (*true positives* as variables and computes the correlation coefficient of both of

them. [23] The MCC value can range from $[-1, +1]$, 1 represents the perfect classifier, -1 represents the perfect mis-classifier (always classify the points as the wrong class), and $MCC = 0$ is the value for classifier with "coin tossing" behaviour. [5]

$$mcc = \frac{(tp * tn) - (fp * fn)}{\sqrt{(tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)}} \tag{4.6}$$

Figure 4.3: Formula for Matthews Correlation Coefficient (MCC)

Since the F1 score is not the optimal metric for comparing the different anomaly detection algorithms, MCC is used for the evaluation, and a particular focus is placed on precision since false positives play a more significant role in the application scenario.

4.2 Non-Functional Criteria

The goal of defining non-function criteria is to estimate the usability of the anomaly detection algorithm in the application in the industry.

4.2.1 Configuration & Setup

For the industrial use of anomaly detection algorithms, the time required for commissioning and the effort required for maintenance are essential metrics. Therefore, anomaly detection algorithms that have simple, well-documented interfaces are preferred. The software code of the algorithm should be published in a well-known programming language, open-source, well maintained, and have an active repository.

4.2.2 Tuning Options

Almost all anomaly detection algorithms need input parameters that change the behaviour or adapt the algorithm to the dataset used for training and testing. A large number of configurable input parameters for an algorithm can be both an advantage and a disadvantage. An advantage can be the finer adaptation to the data set, which in the best case leads to better results in anomaly detection. A disadvantage can be the time required to find the correct parameters, primarily if the data set is unknown or constantly changing. This disadvantage can increase if the parameters are mandatory.

4.2.3 User-related application and acceptance

When putting the monitoring system into production, anomaly detection needs to be combined with alerting the users of the monitoring system. From a DevOps perspective,

the monitoring system should not be continuously observed by the users but only when problems occur. In most applications, the monitoring system is supplemented with an alerting concept that uses threshold values to detect critical application or system states. Ideally, anomaly detection supports the detection of critical states by alerting in time before the state occurs.

For better user acceptance, a higher false negative rate and a lower false positive rate is the goal. Too many false positives will make the user tired because if the systems are checked too often for faulty anomalies, the user will probably assume that future anomaly alerts are not real anomalies, and thus he will check the system no longer regularly. This behaviour leads to the diminishing of the added value of the anomaly detection system. A proportionally higher false-negative classification, on the other hand, does not reduce the added value of the solution, as the existing mode of operation is not affected if no anomalies are detected (reacting to problems vs. early actions).

5 FADIP - Flexible Anomaly Detection Integration Platform

5.1 Requirements

For a valid comparison of different anomaly detection frameworks and algorithms, we built FADIP, the flexible anomaly detection integration platform. FADIP needs to fulfill several requirements.

These requirements were collected from different employees working from a known german car manufacturer, from feedback from university students and from previous experience. Table 5.1 shows the collected non-functional requirements. The functional requirements are shown in table 5.2. Later in this chapter, the requirements are closely reviewed for compliance.

#	Requirement	Description
1	Low environmental impact	The system needs to be able to handle extensive loads of monitoring data without breaking or putting too much pressure on the current monitoring stack
2	Stable & fail-safe	The system needs to be stable and fail-safe so that individual problems allow further operation and malfunction stop damage
3	Easy extensible	The system needs to be easily extensible by adding new anomaly detection algorithms and new datasets.
4	Automatic execution	Training, evaluation, and prediction should be made by the system automatically with no manual interaction involved.
5	Convenient configuration	It should be possible to configure the system easily without deep technical knowledge

Table 5.1: Non-Functional requirements of FADIP platform

#	Requirement	Description
1	Data loading from Prometheus	Time-series data needs to be loaded from a Prometheus or PromQL-compatible service
2	File-based dataset loading	Datasets from files with different formats need to be loaded
3	Anomaly Detection on time-series	On the loaded data, anomalies or outliers need to be identified by different anomaly detection algorithms
4	Alerting & Notifications	For the found anomalies, the user should be able to be notified with more information on the outliers via Slack or Microsoft Teams
5	Evaluation of anomaly detection algorithms	The integrated anomaly detection algorithms should be evaluated with selected datasets, and the results analyzed
6	Persistence of found anomalies	The found anomalies should be persisted together with the time-series identifiers, as well as some additional information
7	Configuration via YAML file	The user should be able to configure the systems, connections, and attributes via a YAML configuration file
8	Triggering via REST API	The different functionalities should be triggered by using an open REST API
9	Local & remote model storage	The models that were generated in the training phase of the anomaly detection should be persisted in the local or remote storage

Table 5.2: Feature requirements of FADIP platform

5.2 Overall architecture

Figure 5.1 shows the internal structure of FADIP and the integration into the IT infrastructure. On the left side, an abstraction of the monitoring stack used in combination with FADIP is shown. The monitoring stack is explained in more detail in chapter 2.

FADIP supports two operating modes, *evaluation* and *detection*. Both operating modes can be triggered through a REST API, but more details about the REST API and the available routes will follow later. For the *detection* mode, at least one PromQL-compatible datasource is required, but multiple are allowed. In figure 5.1, there is one datasource shown, Thanos Query, which is a datasource that can collect metrics from multiple Prometheus instances and multiple object storages by utilizing other Thanos components. On the right side, the communication channels with the users

of FADIP are shown. FADIP supports Slack and Microsoft Teams as communication channels. The center shows FADIP and the different components. The left side shows the data preprocessing and transformation required for all algorithms. The center shows the training and evaluation module, both with the implemented anomaly detection algorithms. The right side shows the component responsible for the result visualization and mapping of results to datasets and sending the visually-understandable results to the configured communication channels. The bottom part shows the component management and the REST API, which handles the users' requests as well as reading the configuration file. It also manages the connection to a PostgreSQL database. On the bottom, outside of FADIP, two other input options are mentioned. First, the potential open-source datasets that need to be placed inside the FADIP platform and the configuration file that needs to be written by the user also placed inside FADIP. Right near the configuration file, another box represents the user that is triggering different functionalities of the platform by sending REST requests to the REST API using a client or an automation tool.

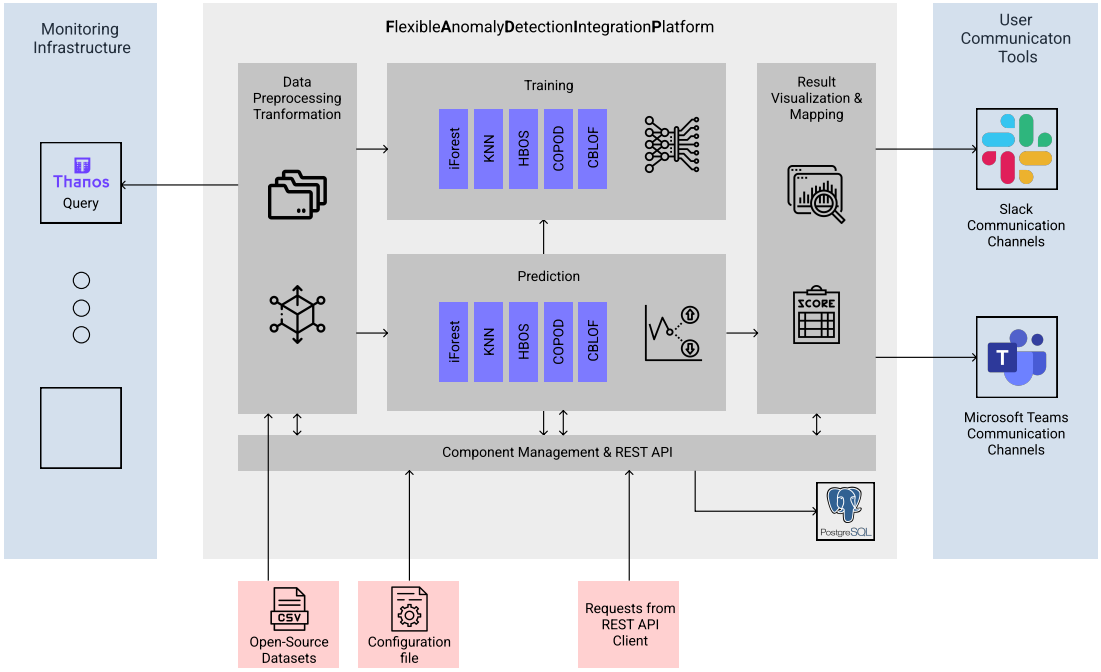


Figure 5.1: Architecture of flexible anomaly detection platform (FADIP)

5.3 Operation modes

As mentioned before, the system supports to operation modes, *evaluation* of anomaly detection algorithms, and continuous *detection / prediction* of anomalies on time-series from Prometheus. First, the general functionality and process are explained, then the differences between evaluation and detection are discussed.

FADIP is a stand-alone Python application packed into a Docker image that can be started (or a standard python application, instructions can be found in the GitHub repository [8]). First, FADIP loads the configuration file and checks the configuration file against a defined yaml scheme. Then, the database credentials are used for trying to establish a connection to a PostgreSQL database. The defined data models are migrated into the database as schemata. The application fails when a non-valid configuration file was used, but a non-existent or failed connection to the database is tolerated. After that, a webserver is opened on port 80, and the REST API is available, a documentation of the REST API is available under http://localhost/docs .

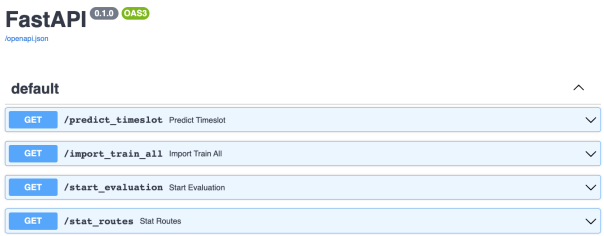


Figure 5.2: Routes exposed by the REST API of FADIP

For the *detection* mode, FADIP provides two of the routes that can be seen in figure 5.2. Continuous prediction of anomalies requires that for every time-series, first, a model with the historical data of the time-series is trained and stored. This model training and creation process can be done by starting a GET request to the route */import_train_all*. The endpoint will return immediately after starting the training process by putting a training task into the background-queue. The Flow-Chart diagram in figure 5.3 explains the scheme of how FADIP trains the algorithms. To understand the algorithm, it is important to know that FADIP works with different for-each loops through the configured datasources, time-series, and algorithms. The user specifies all datasources, then, inside the datasources, the corresponding time-series, and for every time-series the algorithms that need to train with exact this time-series. First, the background task loads the configuration file. Then, for every datasource specified in the configuration, it looks for time-series inside this datasource. For every time-series, it loads the training start-time and end-time and splits this into 24-hour-long parts. Every 24-hour time

interval is then loaded from the datasource, and all time intervals are merged into one time-series. Then, for every algorithm specified for the time-series, a model is fitted with the training data. This model is stored either on the local disk or in an AWS S3 bucket.

FADIPs concept of continuous prediction effectively means that time windows of 2h are constantly examined for anomalies. To analyze the time window for the last 2 hours, a simple request to */start_evaluation* starts the detection of anomalies for all (in the configuration file) specified time-series. The procedure is similar to the training, but only the time window of the last 2 hours is requested from the data source, the existing model is loaded, and then the time series are examined for anomalies with the corresponding model.

In the *evaluation* mode, FADIP is reading datasets, parting them in according to a train-test-split into a training time-series and a testing time-series. An overview of the evaluation process can be seen in Figure 5.4. As in the training process, FADIP first loads the configuration file. Every dataset in the configuration file is loaded and necessary transformations and cleaning are done (preprocessing). In the next step, the dataset is split using the configured *train-test-split* into a training time-series and a testing time-series. If the dataset is labeled, the contamination rate of the training-time-series is calculated using the labels. Then, for each algorithm specified in the configuration file for the current dataset, a model is built using the training time series and algorithm. Then the model is used for a prediction on the testing time series. If the dataset is labeled, the performance of the dataset is evaluated (by the same metrics specified in chapter 4). Finally, the evaluation graphs are built.

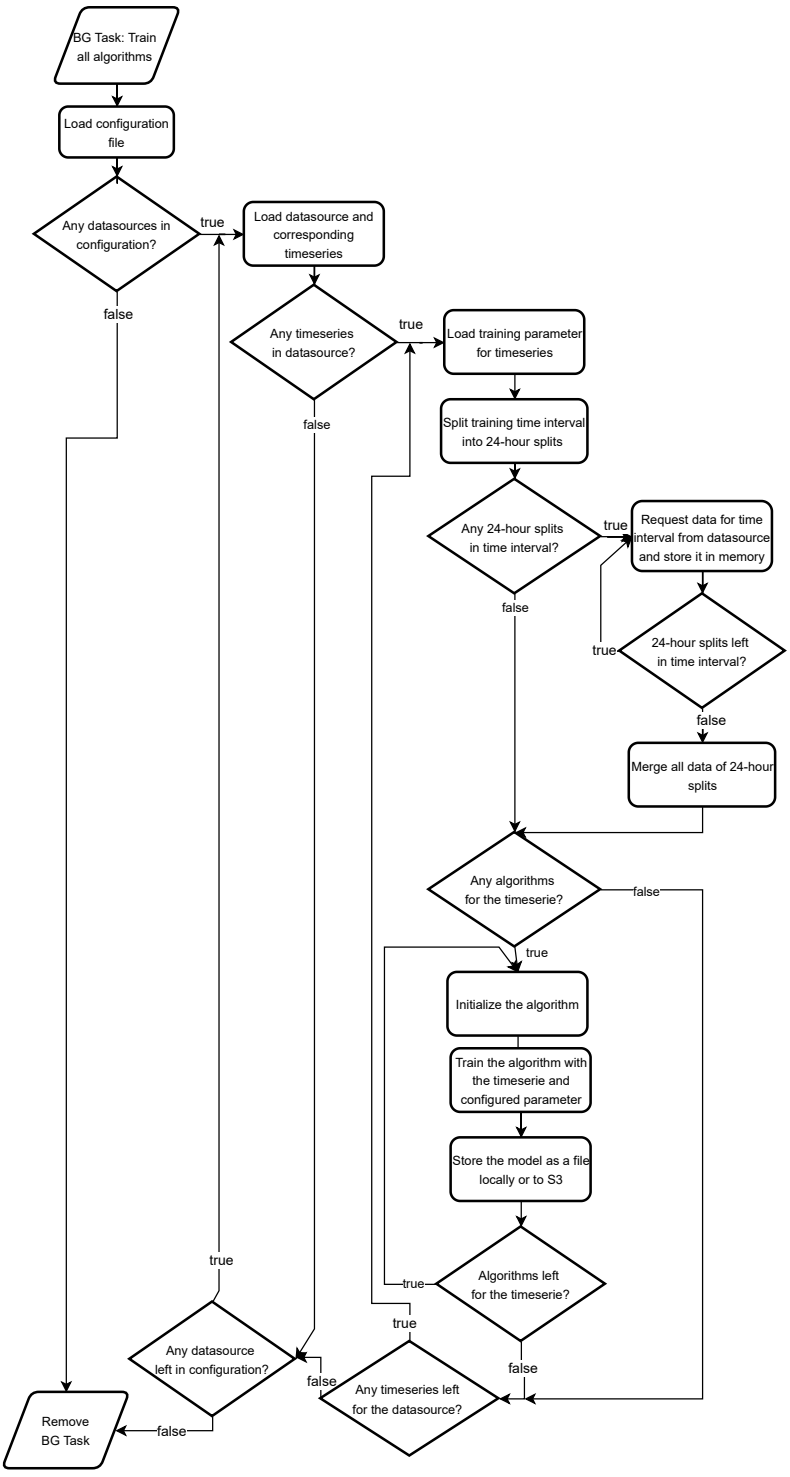


Figure 5.3: Flow-Chart diagram of the training process of FADIP

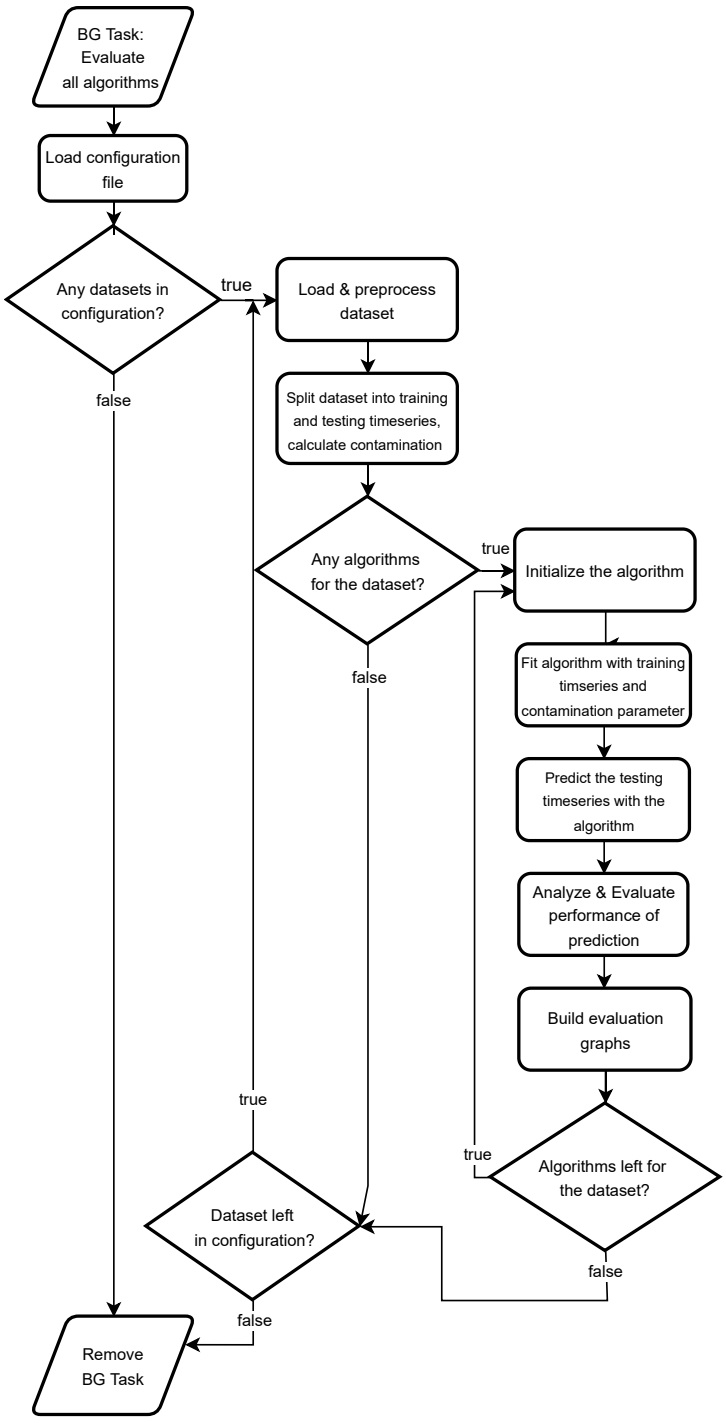


Figure 5.4: Flow-Chart diagram of the evaluation process of FADIP

5.4 Technical Details

In this section, the relevant technical details of the platform are described.

5.4.1 Configuration

FADIP relies on a single configuration file that specifies all involved systems and more. The system hot-reloads the configuration file, which means that the application doesn't have to be restarted and reloads the configuration file every time a REST API request is accepted by the system.

The configuration file consists of several sections:

1. **Datasources**

   In this section, all PromQL-compatible datasources should be specified, for example, different Prometheus datasources. Currently, the only datasource type supported is "prometheus". The id can be freely chosen but need to be unique. It is used for the mapping of time-series to datasources. The application must have access to the given URL; username and password are optional.

2. **Management Database**

   The management database is responsible for storing the evaluation results and the found anomalies for later analysis and usage. The connected database can only be a PostgreSQL database, and hostname, port, username, password, and database need to be specified.

3. **Model Storage**

   In bigger production environments, there is maybe a need to store the models not on the local disk (due to disk storage). Therefore it is necessary to configure credentials to AWS S3 (simple-storage-service) and enable the s3 model storing flag and activating it via the flag (activated flag).

4. **Alerting**

   FADIP supports two different integrations for notifying about potential anomalies, Slack and Microsoft Teams. Both options can be configured there, Slack needs an OAuth Token, and Microsoft Teams needs a webhook URL. Leaving one or both empty leads to missing alerting/notification feature.

5. **Mapping**

   In the mapping section, the user can specify which time-series from which datasources will be analyzed for anomalies. The analysis consists of two steps; the

first is the training (or fitting) phase. In the training phase, FADIP loads the Prometheus data from the specified query and the corresponding Prometheus instance (through the datasource_id) from the training_starttime to the training_enddtime (both formatted as Unix timestamps). The specified algorithms are then trained with the loaded data, and the model file is stored either on the local file system or in AWS S3. For every algorithm, also a training parameter can be specified.

The second step is the prediction of anomalies with the newly created models. This is also triggered by a REST API request.

6. **Evaluation**

   The evaluation section allows the user to configure the evaluation of different open-source datasets with the different anomaly detection algorithms. The section allows defining the output directory of the created analysis graphs as well as an output directory of the raw dataframes as pickled objects. For every dataset, a unique id has to be given, together with the local path for the dataset. Also, there is a need to specify time-series-type, either univariate or multivariate and whether the dataset is labeled or not. For every dataset, the different algorithms can be specified, with their identifier, the *train-test-split* (train_percentage), and optionally the *contamination_train* which is used as input parameter for the training phase of the algorithm. First, the dataset is split by the factor specified in the *contamination_train*, and then, the first part is used for the training phase, and the other part is used in the testing phase. The algorithm's outcome is then compared to the labels (if the dataset is labeled) and evaluated after several parameters and the results stored in the management database.

```
fadip:
  version: 0.1
  inital_setup: True
  working_mode: "normal"
  datasources:
    - type: prometheus
      id: "prom1"
      url: "http://0.0.0.0:9090"
      username: ""
      password: ""
  management_database:
    host: "localhost"
    port: "5432"
    username: "*******"
    password: "*******"
    db_name: "flexadf"
  model_storage:
    s3:
      aws_access_key_id: "******************"
      aws_secret_access_key: "****************************************"
      bucket_name: "masterthesis-ad"
      activated: True
  alerting:
    slack:
      oauth_token: "***************************"
```

```yaml
    teams:
      webhook_url: "https://**********"
  mapping:
    - datasource_id: "prom1"
      time-series:
        - id: "localhost_process_cpu_seconds"
          query: "rate(process_cpu_seconds_total{instance='localhost:9090', job='prometheus'}[5m])"
          chunk_size: "5m"
          algorithms:
            - id: "iforest"
              contamination_train: 0.05
            - id: "copod"
              contamination_train: 0.02
          alerting: false # | false
          training_starttime: 1637587367
          training_endtime: 1637593905
          ts_type: "univariate"
        - id: "cadvisor_process_cpu_seconds"
          query: "rate(process_cpu_seconds_total{job='prometheus'}[5m])"
          chunk_size: "5m"
          algorithms:
            - id: "knn"
              contamination_train: 0.01
            - id: "cblof"
              contamination_train: 0.01
          alerting: false # | false
          training_starttime: 1637587367
          training_endtime: 1637593905
          ts_type: "multivariate"
    - datasource_id: "prom2"
      time-series:
        - id: "prom2_localhost_p_c_s"
          query: "rate(process_cpu_seconds_total[5m])"
          chunk_size: "5m"
          algorithms:
            - id: "hbos"
              contamination_train: 0.02
          alerting: false
          training_starttime: 1639044894
          training_endtime: 1639064793
          ts_type: "multivariate"
  evaluation:
    graph_output_dir: "../evaluations/graphs/"
    df_output_dir: "../evaluations/raw_data/"
    datasets:
      - id: "ms_middle-tier-api-dependency-latency-uv"
        local_path: "./datasets/MSCloudMonitoringDatasets/data/middle-tier-api-dependency-latency/outbound_2x-01.csv"
        ts_type: "univariate"
        labeled: true
        unsupervised: false
        algorithms:
          - id: "copod"
            train_percentage: 0.5
            contamination_train: 0.001
          - id: "loda"
            train_percentage: 0.5
      - id: "ms_data-ingress-rate-05"
        local_path: "./datasets/MSCloudMonitoringDatasets/data/data-ingress-rate/ingress-05.csv"
        ts_type: "univariate"
        labeled: true
        unsupervised: true
        algorithms:
          - id: "knn"
            train_percentage: 0.5
          - id: "cblof"
            train_percentage: 0.5
```

5.4.2 Frameworks

The system relies on several open-source projects to fulfill the functional and non-functional requirements. The basis of the implementation is a Python *FastAPI* application server that exposes several routes to consumers. The Python application is deployed using an *uWSGI* server (to fulfill the production requirements of a server) inside a Docker container. The image of the Docker container is based on *tiangolo/uvicorn-gunicorn-fastapi:python3.8*, and the application is running on port 80 of the container.

For working with the different datasets, the known data-science library *pandas* is used, and the incoming data is transformed into *pandas* dataframes. To receive the data from the Prometheus instance (and or PromQL-compatible datasource), the library *prometheus-pandas* reads the data from a specified URL and transforms it into dataframes. To store the results of the evaluation and the detected anomalies in a system, a PostgreSQL database is used, and to connect to this database, *SQLAlchemy* (Object-Relational-Mapper) with *psycopg2* as a database driver is used. When it comes to the actual anomaly/outlier detection, several algorithms were used, mostly already existing implementations taken from the *PyOD* library [37] here. For a detailed analysis, the plots of the time-series, especially with the important areas, are visualized using the *matplotlib* library.

The application is packaged into a Docker image and can be started as a container to allow a smooth deployment. Also, a pre-configured docker-compose is available that starts a PostgreSQL database, a Prometheus Instance, and the FADIP application. Docker was chosen as a virtualization platform because of its advantages, it allows the code to be portable, and the system can be deployed on almost any other system or platform. Docker is also more performant than other virtualization methods, and it is possible to scale the platform horizontally (by running multiple instances of the FADIP platform), more on the scalability of FADIP later. [3]

5.4.3 Data model

Figure 5.5 shows the data model used internally and in the database. The model is written in code, and SQLAlchemy transforms it into a PostgreSQL-compatible database scheme. The model includes two tables, *anomalies* and *evaluation*. Anomalies store some statistical data as well as important information that make the anomaly traceable. The stored data contains, among other fields, the query that was used to retrieve the data from the datasource, and the used algorithm, the date when the anomaly was identified. *start_investigated_datetime* and *end_investigated_datetime* are the starting and ending date of the excerpt of time-series that was analyzed for anomalies, and if the algorithm found some, the date of the first and the last anomalous datapoint are stored

in *first_anomaly_datetime* and *last_anomaly_datetime*. Evaluation table holds the results of the performance evaluations of the different anomaly detection algorithms. The data includes the exact dataset, whether it is uni- or multivariate data, and different statistical like the *true_positives*, *true_negatives*, *false_positives*, *false_negatives*, *MCC*, and more.

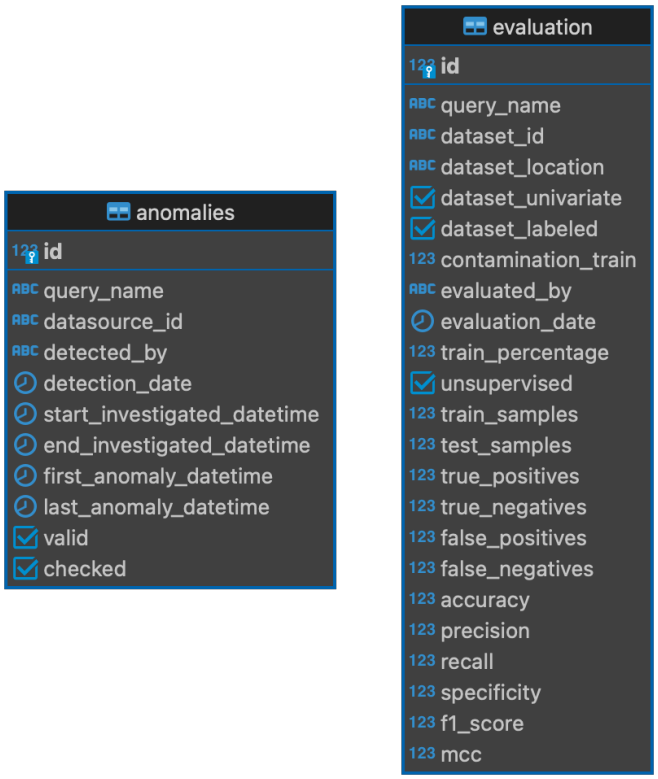


Figure 5.5: Entity-Relationship diagram of FADIP data model

5.5 Requirements Compliance

In general, the functional requirements were all fulfilled by the features implemented in the platform, so in this section, the focus lays on the non-functional requirements and their fulfillment.

5.5.1 Low environmental impact

The goal of this non-functional requirement was mainly to let the FADIP application operate without any impact on the current existing platform. To achieve this requirement, the data loading process from the monitoring stack was optimized. The performance of queries in Prometheus and PromQL-compatible time-series stores depends on a number of factors, such as the scrape interval, the amount of features, the labels of the time-series, the functions applied in the query and more. In summary, however, personal experience and the architecture of Prometheus and Prometheus-compatible tools show that the CPU and memory of queries that cover a longer period of time and contain several features can affect the performance of the monitoring system.

To solve this problem, FADIP uses a method to reduce the peak load to a continuous load under an extension of the data loading time. For both the training and prediction phase, the data loading method splits the timeframe of every query into "dataframes" by days so that every "dataframe" is at maximum 24 hours long and requests these dataframes sequentially. When different time-series are requested, the single time-series is requested (using the method mentioned the in sentence before), and then immediately the algorithms analyze the data, and after analysis, the next time-series is requested. This also leads to pauses in the requests to the monitoring stack so that the system can handle other requests & loads.

5.5.2 Stable & fail-safe

System stability and fail-safe behaviour are equivalently important for FADIP. Both are achieved by different methods. First of all, the application can be at any point restarted because it is completely stateless, and all requests are handled in the background-queue, so the system cannot switch into a state like *stuck* or *need for manual interaction*. Also, the system can handle failing or wrongly implemented anomaly detection algorithms. It informs the user about the problem but continuous its operation with the following algorithms. The same applies to missing datasources, requests for time-series with timeouts, and requests for wrong timespans of monitoring data. Also, downtimes of other involved systems are ignored, FADIP can ignore a potential database failure (but does not cache the detected anomalies or evaluations), it can ignore problems with single or multiple datasources, and potential down-times & misconfiguration of the alerting and notification tools.

5.5.3 Easier extensible

System extension should be as easy as possible. This requirement is fulfilled by the overall design & architecture. The extension of datasets for evaluation is easy; the

configuration file allows to specify the dataset and the code marks an entry-point for the needed transformations. The dataset is identified by a freely selectable identifier and the target transformations that are needed to be performed for the algorithms that are specified in the code. Algorithms can be easily added be just building another python class that inherits the predefined functions from a meta-class, and the transformation and mapping of the different methods is done in the algorithm class. The metaclass defines only the functions: *train_algorithm_unsupervised*, *predict_sample*, *load_model_from_file*, *store_model_to_file*, *store_model_to_s3* and *load_model_from_s3*. All added algorithms must implement these functions and be compliant to their structure and sense.

5.5.4 Automatic execution

Automatic execution refers to the least-possible human interaction. FADIP should operate at best entirely without human interaction. Since the system is built with a REST-API and background-tasks from FastAPI, the only interaction can be done through the available REST endpoints. The REST endpoints are designed only as triggers so that the trigger interval can be freely chosen. The system's complete autonomous operation is possible using periodic API triggers, implemented using AWS Lambdas or cronjobs. The supposed way is to use AWS Lambda and implement a lambda function that periodically triggers the API. This allows better monitoring of the system.

5.5.5 Convenient configuration

Convenient configuration is achieved through the defined yaml scheme for the configuration. The given configuration file can be changed, and while the application is running replaced, and FADIP automatically reloads. The configuration file is also validated against a scheme, so the application does not start with an invalid configuration.

6 Evaluation

6.1 Datasets

For a good evaluation of the different anomaly detection algorithms and frameworks, this thesis includes different anomaly-detection datasets. Since this thesis focuses on anomaly detection for time-series data from car monitoring, we only consider the technically similar or somehow related time-series from all selected datasets.

6.1.1 Microsoft Cloud Monitoring Dataset

Microsoft, Inc published a real-world cloud monitoring dataset from their services and client telemetry signals. The dataset is a collection of 9 time-series, each having three features: TimeStamp, Value, and Label. The TimeStamp represents the exact time at which the datapoint was collected, the Value is the value of the metric that is watched by a single time-series, and the label designates whether a datapoint is considered as an anomaly, with 1 equals an *anomaly*, and 0 *normal*. [24]

In this thesis, we performed the evaluation on the following datasets:

Time-series Name	**Description**
data-ingress-rate	Data ingress rate into a cloud service, with different variations and service disruptions.
ecommerce-api-incoming-rps	Incoming request rates to an ecommerce api
application-crash-rate-1	Represents unusual increases in application crash rates relative to baseline, normalized to application launch counts
application-crash-rate-2	Represents unusual increases in application crash rates relative to baseline, normalized to application launch counts

Description out of the metadata.csv files from the respective directories.

Dataset: application-crash-rate-1

This dataset shows the unusual increases in application crash rates relative to the baseline, normalized by the number of application launch counts. This dataset contains

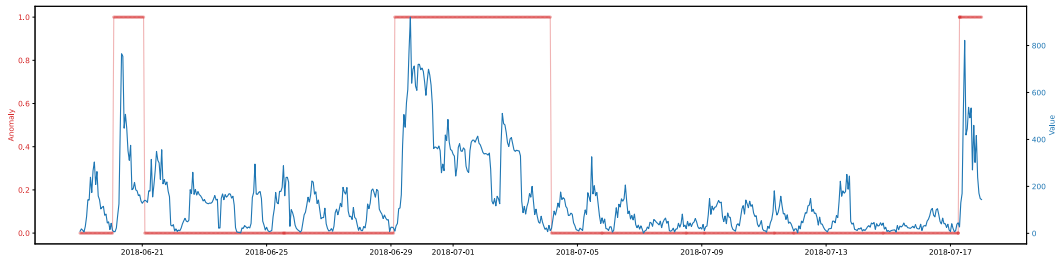


Figure 6.1: Timeserie: application-crash-rate-1: app1-03.csv

9 different time-series inside CSV files, all with different characteristics and amounts of anomalies. Three datasets were selected due to their characteristics, the corresponding CSV files are: *app1-01.csv*, *app1-03.csv*, *app1-07.csv*. An example dataset, which can be seen in Figure 6.1 shows the CSV file *app-03.csv*. The time-series shows no significant trend over time. It contains seasonal patterns, but they do not follow an exact scheme with minimal derivations. They exist outside the areas marked as anomalies, with different amplitudes, but mostly in the same wide. This time-series contains a high rate of anomalies; when looking at the first 80% of the dataset (used for training the algorithms), the percentage of anomalies is 25%. This dataset was chosen because it comes close to monitoring an application cluster (e.g., several scalable services in a Kubernetes cluster), where services are terminated or crashed (e.g., in the event of a DDoS attack) and are restarted.

Dataset: application-crash-rate-2

The dataset *application-crash-rate-2* contains 10 CSV files with each a time-series inside. As dataset application-crash-rate-1, it displays the application crash rates relative to the baseline normalized by the number of application launch counts. Out of the 10 time-series, 2 were selected due to their characteristics, *app2-01.csv* and *app2-09.csv*. The time-series in Figure 6.2 was selected due to the (in comparison to other time-series from the dataset application-crash-rate) high amplitude in the seasonal sections and because of the few anomalous points in the whole set. When using the first 80% of the dataset for training and the remaining 20% for evaluation, both parts only contain one "section" with anomalous datapoints. This dataset was chosen because it comes close to monitoring an application cluster (e.g., several scalable services in a Kubernetes cluster), where services are terminated or crashed (e.g., in the event of a DDoS attack).

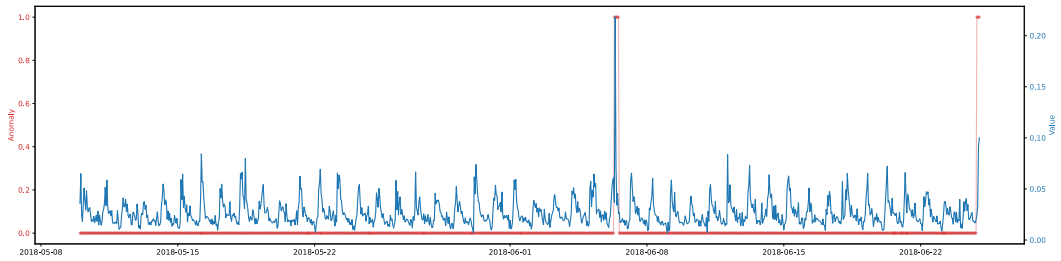


Figure 6.2: Timeserie: application-crash-rate-2: app2-09.csv

Dataset: data-ingress-rate

This dataset includes datapoints of the data ingress rate of an anonymous cloud service. Figure 6.3 and figure 6.4 shows two uni-variate time-series graphs. In both graphs, *Value* and *Label* are plotted, with datapoints starting at 05.04.2018 and ending at 05.05.2018. Out of the provided open-source dataset from Microsoft, only the files: *ingress-01.csv*, *ingress-04.csv*, *ingress-05.csv* were found to be useful for the evaluation. All three time-series show two relatively big (in comparison to the whole dataset) anomalous sections in the same time area. These two anomalous sections are drops in the data-ingress with varying amplitude. *ingress-04.csv* and *ingress-05.csv* add additional anomalies, in all cases high deflections in the graphs. This dataset was chosen because it contains very similar metric data to a typical backend service in the automotive industry, for example, the metrics of calls to the weather service by the infotainment system installed in the vehicle.

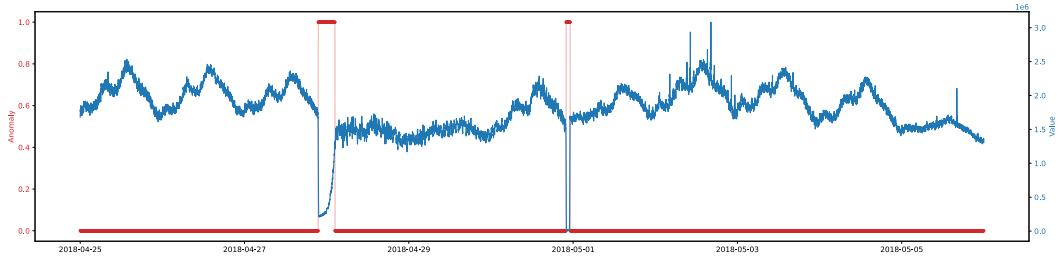


Figure 6.3: Time-series data-ingress-rate: data-ingress-01.csv

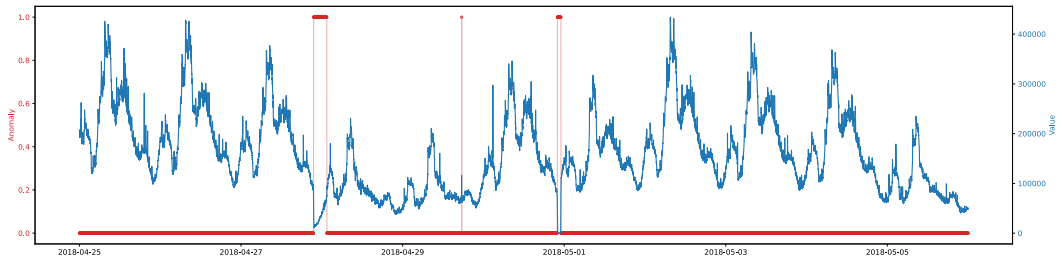


Figure 6.4: Time-series: data-ingress-rate: data-ingress-04.csv

Dataset: ecommerce-api-incoming-rps

This dataset shows the incoming request rate on an e-commerce API. Figure 6.5 shows the plot of the dataset values (in blue) and the points designated as anomalies (in red). The labeled anomalous are, without exception, peaks in the number of requests. While the periodicity is very narrow, there is no discernible trend in the values. The anomalies are almost evenly distributed. This dataset can be compared in particular with the metrics of an online shop for parking space bookings of an infotainment system in the vehicle.

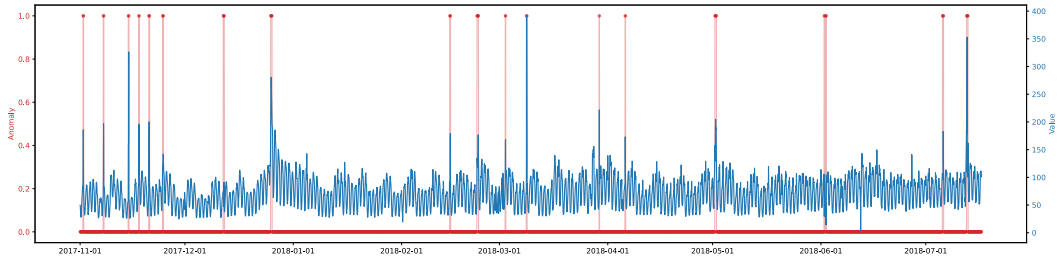


Figure 6.5: Time-series: ecommerce-api-incoming-rps: api-01.csv

6.1.2 Anonymized commercial datasets

This section presents the evaluated anonymized commercial datasets from a known german car manufacturer. The dataset represents the sum of the established connections to an application delivery controller but separated by regions and the target application of the request. The regions in the dataset are: *reg1* and *reg2*, and the selected applications are *app4*, *app8*, *app10*, *app17* and *app18*, however, not every application is available in every region. The available combinations and datasets used are shown below. Almost all time-series in this dataset contain daily and weekly seasonalities, which is due to the fact that the average car infotainment usage correlates with the average car usage of

the population (drive to work in the morning and back in the evening). All time-series in the datasets were sampled from data points every 15 seconds to data points every minute to improve performance, and because such fine-granular data is not necessarily more meaningful for monitoring applications, this downsampling often occurs when data is stored for long periods.

Application 4

When looking at the dataset in figure 6.6, it is relatively straightforward that there is a weekly as well as a daily seasonality. There is no discernible trend visible, but a few anomalous datapoints are visible, especially in the middle and at the beginning of the second half of the time-series. There is also a temporary change in the characteristic pattern of the time-series.

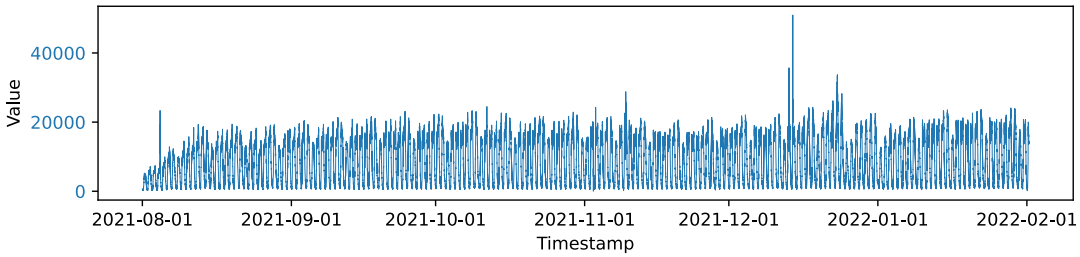


Figure 6.6: Time-series: Region 1 - Application 4 (reg1-app4.csv)

Application 8

Application 8 is available in two regions, region 1, and region 2. The two time-series in figure 6.7 and in figure 6.8 differ in their details, except for one common feature. This common feature is that both time-series show a low amplitude in their daily and weekly seasonalities from the halfway point onwards. This makes both time-series particularly interesting, as there are anomalies in both, and the pattern changes seasonalities, i.e., a certain trend is present, but it is not an anomalous behaviour. Both time-series contain anomalies, so *train-test-splits* of 0.8 or more should not result in detected anomalies.

Application 17

Application 17 is also available in two regions, region 1 and region 2. The two time-series in figure 6.9 and in figure 6.10 are fundamentally different, while the time-series of region 1 show a continuously rising trend with daily and weekly seasonality. There are a

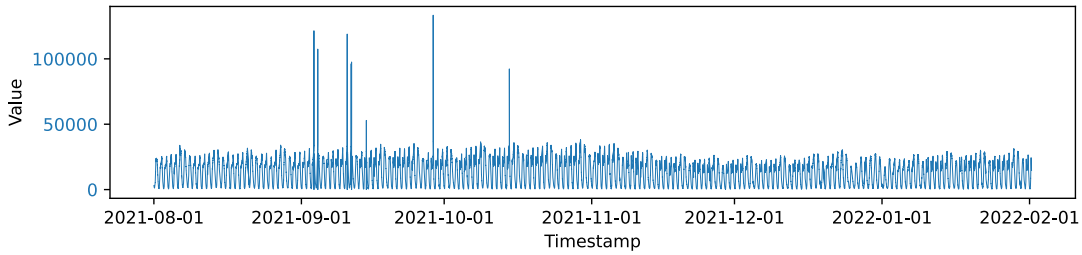


Figure 6.7: Time-series: Region 1 - Application 8 (reg1-app8.csv)

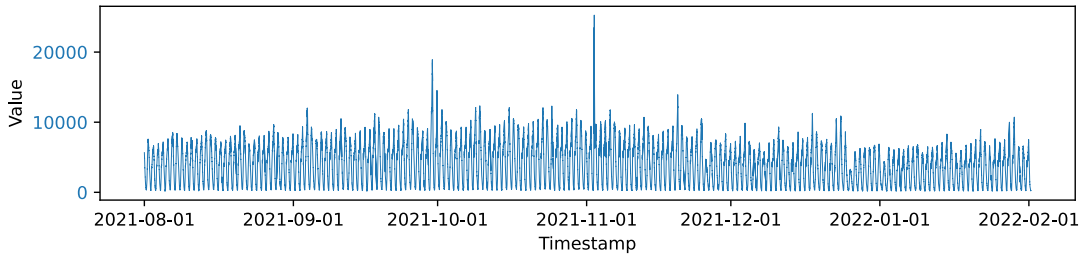


Figure 6.8: Time-series: Region 2 - Application 8 (reg2-app8.csv)

few peaks in the dataset, and the peaks are continuously rising and have an increasingly bigger amplitude. In contrast, the other time-series has no well recognisable seasonality, and no trend, but some enormous spikes (outliers) that are mostly concentrated in the middle of the time-series.

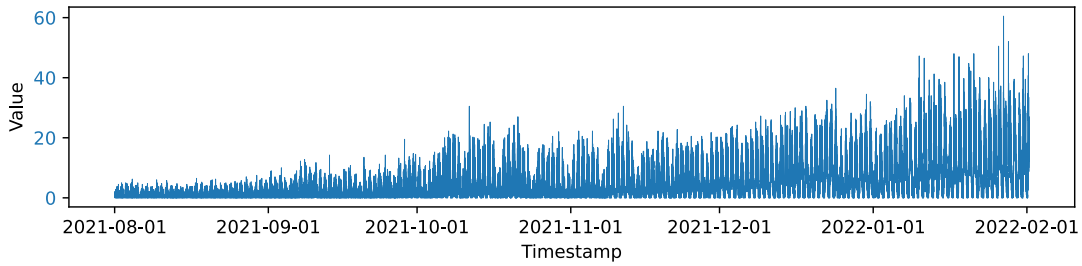


Figure 6.9: Time-series: Region 1 - Application 17 (reg1-app17.csv)

Application 18

Application 18 of region 2 was selected because it is an irregular time-series with seasonality but constantly varying amplitudes and some anomalous points in the

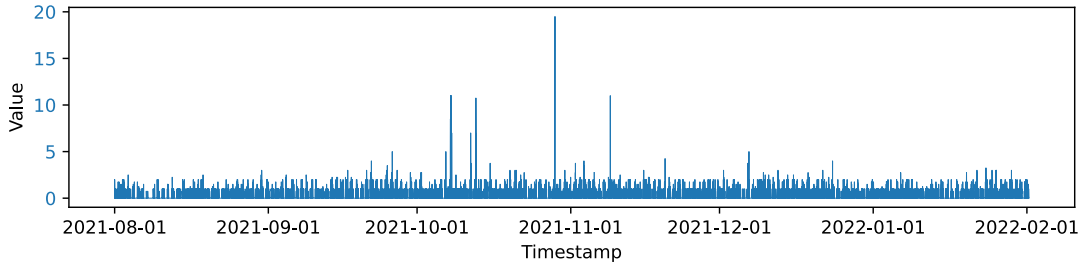


Figure 6.10: Time-series: Region 2 - Application 17 (reg2-app17.csv)

beginning. However, after the first month of the time-series, there are no anomalies that an algorithm could detect.

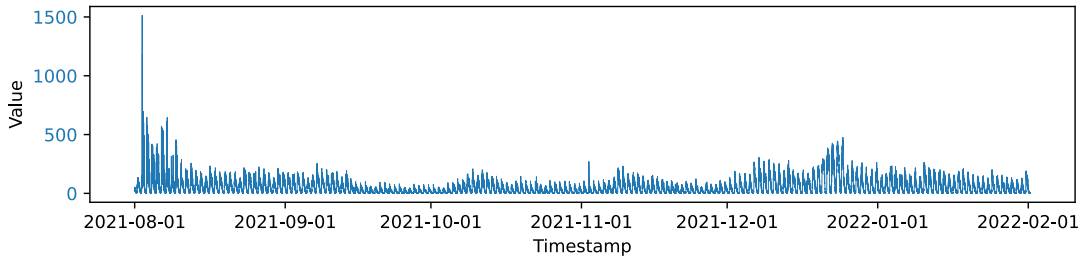


Figure 6.11: Time-series: Region 2 - Application 18 (reg2-app18.csv)

6.2 Algorithm performance on datasets

In this chapter, the selected anomaly detection algorithms explained in chapter 2 are compared and evaluated. The configuration file necessary for the evaluations can be found in this Github repo [8].

6.2.1 Outlier detection on the Microsoft Cloud Monitoring Dataset

When looking at the open-sourced cloud monitoring dataset from Microsoft, we selected different time-series. In order to compare the algorithms fairly, the datasets described in this chapter before are divided into training & test data. This split is represented by the *train-test-split*, where a *train-test-split* of 80% (or 0.8) means that the 80% of the dataset is used for training and the remaining data for prediction. The prediction is then compared with the true anomalies (indicated by labels), and the metrics are calculated.

In this section, the anomaly detection algorithms are compared by some metrics based on the different *train-test-splits* on the single time-series.

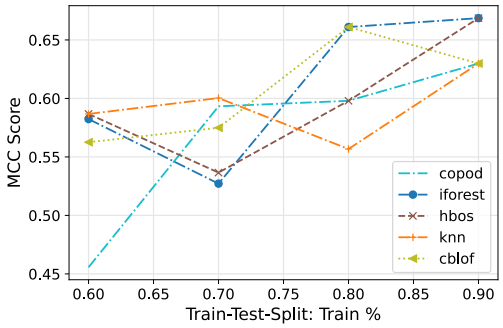
Dataset: Application-Crash-Rate

Figure 6.12 shows the MCC and specificity scores of all algorithms for different *train-test-splits* on the dataset *application-crash-rate1*. Looking at the MCC scores of the algorithms on time-series 6.12a, the algorithms do not behave consistently. While iForest and HBOS have a relatively good MCC score for 0.6, the MCC score drops for a *train-test-split* of 0.7 and after that increases to levels above from the values for 0.6 and 0.4. COPOD shows the MCC scores' expected behavior with the increase of the proportion of training data. KNN and CBLOF behave relatively erratically, increasing for some *train-test-splits* and decreasing again for other *train-test-splits*. An explanation of this behaviour can be found in figure 6.12b, which shows that the amount of false positives in the detections of the algorithms continuously increases (and the specificity decreases). Interestingly, KNN has the maximum of false-positives with a *train-test-split* of 0.7 and KNN with a *train-test-split* of 0.8. The only valid explanation for the differences in the performance of the algorithms is the sensitivity of the contamination rate (that changes with the *train-test-split*) and the characteristics of the algorithm.

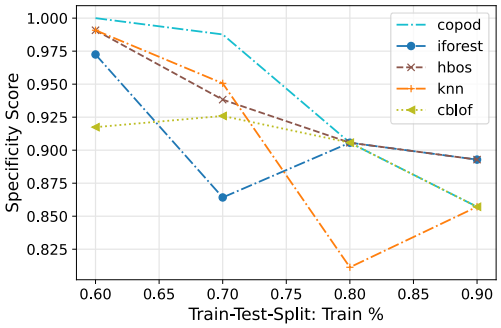
On the time-series app1-07.csv, figure 6.12c shows the expected behaviour that for almost all algorithms, more training data leads to better performance (except for a minimal difference at a *train-test-split* of 0.7 and 0.8 for HBOS). The specificity scores in figure 6.12d show that for KNN, HBOS, and CBLOF the amount of false-positives is relatively high at *train-test-split* of 0.8, however the MCC score is increasing for this *train-test-split*. One explanation for this would be that the algorithms get preciser on detecting the actual anomalous datapoints and leave more false-positives out.

In the time-series plotted in 6.12e, the MCC value of the COPOD algorithm is significantly lower than the values of the 4 other algorithms (which are all on the same MCC curve with the same values). The reason for the equal performance of KNN, HBOS, iForest, and CBLOF on the time-series can be explained by looking at the evaluation of one of the algorithms on the test-time-series 6.13. The test-time-series contains a small range of anomaly points towards the end of the time-series that are significantly higher than the rest of the data. Looking at the entire time-series, it is clear that the time-series is very well suited for anomaly detection, as the anomalies are extreme peaks compared to the normal datapoints. Looking at the classification of the 4 algorithms in figure 6.13, it is clear that the most significant deflections were detected as anomalies, but not the point immediately before and immediately after. These two points are, however, noted as anomalies in the given dataset from Microsoft; if correct, the performance of the algorithm would be equal to that of a perfect detector (and

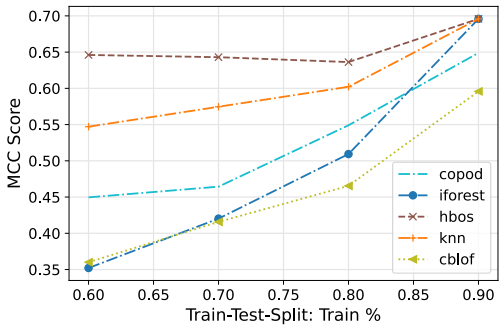
the MCC score = 1) The unusual behaviour of COPOD on the time-series 6.12e cannot be explained, but it can be observed that the only difference with a *train-test-split* of 0.6 is one point less of false negatives and one point more of true positives, for the actual application this difference is not relevant, and the performance of all algorithms is sufficient for a detection (since no algorithm produces false positives, which can be seen in figure 6.12f).



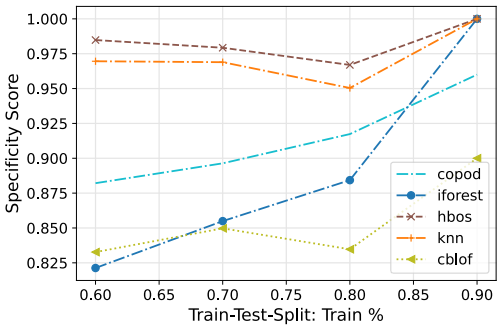
(a) MCC of time-series: app1-01.csv



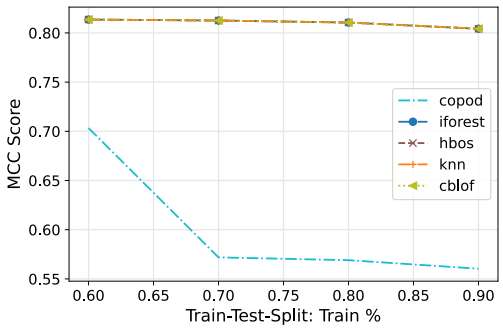
(b) Specificity of time-series: app1-03.csv



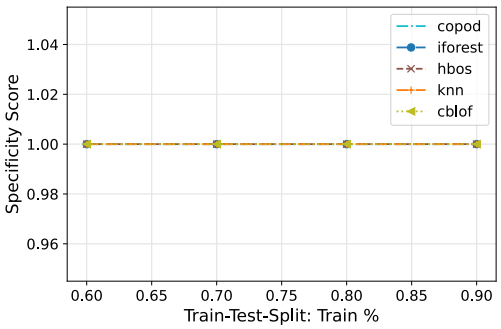
(c) MCC of time-series: app1-03.csv



(d) Specificity of time-series: app1-03.csv



(e) MCC of time-series: app1-07.csv



(f) Specificity of time-series: app1-07.csv

Figure 6.12: MCC & specificity scores of all algorithms on application-crash-rate1 dataset

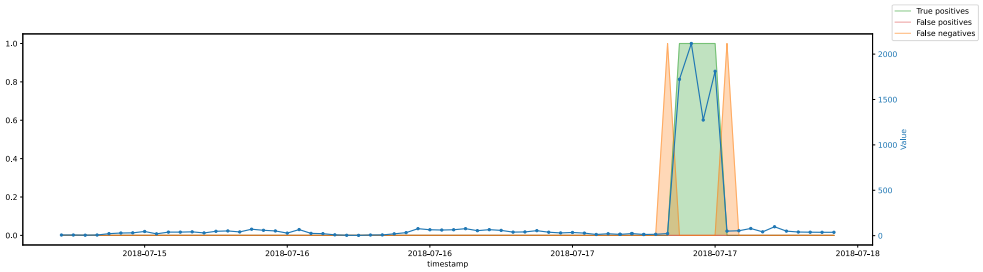
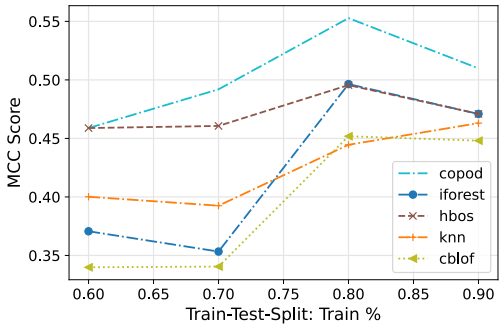
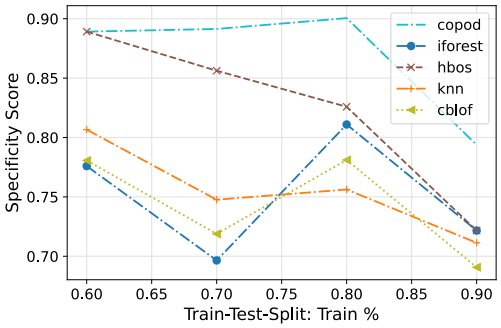


Figure 6.13: Evaluation result of app1-07.csv with HBOS

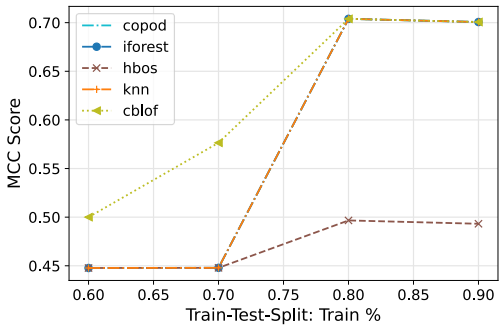
Looking at the other two time-series of the application-crash-rate2 dataset 6.14, the performance of all algorithms is the best for a *train-test-split* of 0.8 (except KNN, which achieves a little better performance in 0.9, as seen in figure 6.14a). Interestingly, for a *train-test-split* of 0.7 there were more false-positives as for 0.6 while having almost a constant MCC score, and the same behaviour for 0.9 in relation to 0.8. On this dataset, COPOD performs overall best since, for app2-01.csv, COPOD always achieves the highest MCC score and highest specificity (lowest amount of false-positives). When looking at the time-series of app2-09.csv, all algorithms except CBLOF stay on the same curve in the specificity curve 6.14d, and all algorithms reach a specificity score of 1.0 for *train-test-splits* of 0.8 and above. The worst performing algorithm on this dataset is HBOS, for *train-test-splits* of 0.8 and 0.9 (figure 6.14c), the MCC score is fundamentally lower. The algorithm's performance on this dataset can again be only explained by the different sensitivity of the contamination rate and the shift of anomaly points from the test time-series to the train time-series with increasing train-test-split. The time-series app2-09.csv is also interesting because it only includes one section of anomalous datapoints in the middle and one in the section that needs to be predicted.



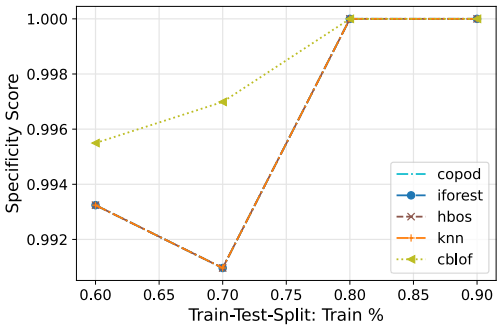
(a) MCC of time-series: app2-01.csv



(b) Specificity of time-series: app2-01.csv



(c) MCC of time-series: app2-09.csv



(d) Specificity of time-series: app2-09.csv

Figure 6.14: MCC & specificity scores of all algorithms on application-crash-rate2 dataset

Dataset: Data-Ingress-Rate

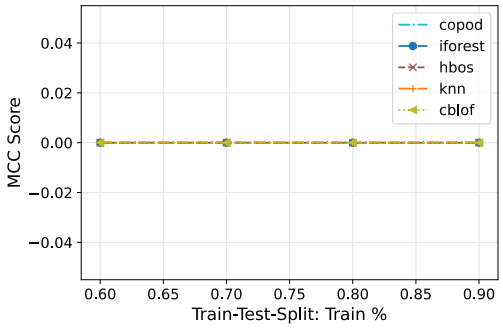
Looking at the time-series from *ingress-01.csv* the MCC score is for all *train-test-split* from 0.6 to 0.9 not available since there are no anomalies in the test data set (figure 6.15a). When 0.4 or 0.5 is selected as the *train-test-split*, there is an MCC score available, as there are anomalous points in the dataset. Since there are no anomalies in the time-series, no anomalies are recognisable, false negatives cannot appear, and there should be as few false positives as possible. Specificity would be the next metric that would make sense for a fair comparison. The figure 6.15b shows the specificity of all algorithms with the different *train-test-splits*. The optimal value of the specificity should be 1 for all algorithms. As can be seen in the graph, all algorithms reach the ideal value with a train-test split of 0.9, but they differ fundamentally with lower *train-test-splits*. The algorithms COPOD and KNN stand out here in particular, and there is another

clear difference between the algorithms at 0.6 and 0.7 as a *train-test-split*. At 0.6 and 0.7, there are still some differences due to the lack of representative datapoints, and the best performing algorithm was HBOS. Interestingly, the jump in specificity between 0.6 and 0.7 is the largest for all algorithms. This is due to the fact that there are now fewer individual points in the test data set that are slightly higher, and half of these have been included in the training set when using 0.7 as *train-test-split*.

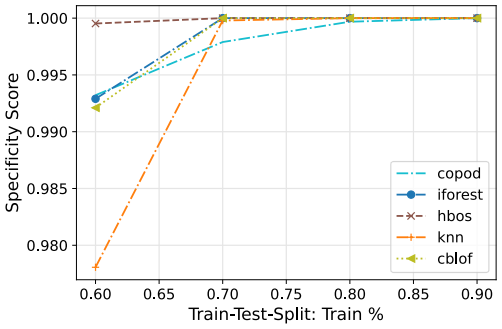
However, an MCC score is available when using 0.4 or 0.5 as *train-test-split*. With 0.4 and 0.5, KNN and COPOD perform worse than all the other algorithms, and HBOS performs best, followed by CBLOF and iForest. The bad performance of KNN could not be improved relevantly by varying some of the input parameters (like the exact detector). The rule: "more training data, fewer test data" led to a better result for all algorithms.

For the time-series *ingress-04.csv* the situation is similar, like ingress-01, ingress-04 also contains no anomaly points for train-test splits in $[0.6, 0.9]$. The time-series differ only slightly in the number of anomalies, but the amplitude of the seasonality of the normal points is significantly higher. Exactly this increased amplitude shows its effect when looking at the number of false positives (respectively the specificity) at low *train-test-splits* in figure 6.15d. The number of false positives in the time-series decreases progressively with the amount of training data. Points that were marked as false positives in all training test splits were no longer marked in the highest training test split of 0.9, which is a very positive effect.

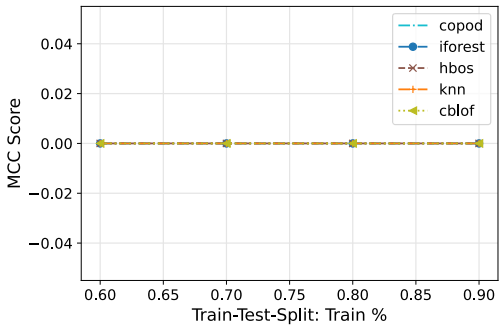
Time-series *ingress-05.csv* is very similar to ingress-rate 01, but the last third of the dataset contains a strong deflection that is recognised as an anomaly point, which is part of the test data series for train-test-slits of $[0.4, 0.7]$. For $[0.8, 0.9]$ as a *train-test-split*, specificity can again be used as a comparative metric, since the number of false positives is decisive here, and for $[0.4, 0.7]$ the MCC score. A closer look at the MCC scores in figure 6.15e reveals a very unusual effect: COPOD achieves enormously high MCC scores on the smaller training test splits 0.4 and 0.5, in contrast to all other algorithms. This effect can only be explained by the amount of data used for training and testing, with a *train-test-split* of 0.5 the test data contain a large section with some anomalies, which were all correctly recognised by COPOD, with 0.6 this section is not present. However, in contrast to all other algorithms, COPOD detected significantly fewer false positives. The specificity scores on the time-series show similar behaviour for iForest, CBLOF, and KNN as in the other time-series from the same dataset; at 0.6 and 0.7 as a *train-test-split*, too many false positives are output, only COPOD and HBOS output few false positives. Overall, COPOD performs best on the time-series ingress-05.csv. HBOS struggles recognising the true positives with smaller *train-test-splits*.



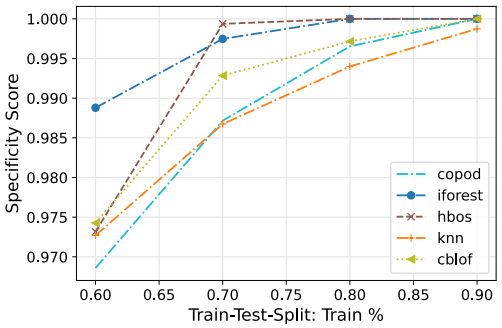
(a) MCC on time-series: ingress-01.csv



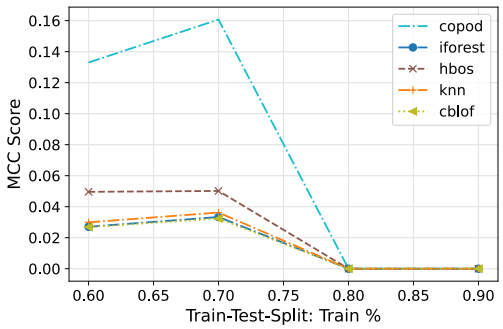
(b) Specificity on time-series: ingress-01.csv



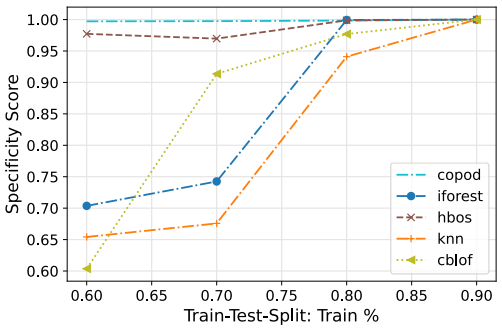
(c) MCC on time-series: ingress-04.csv



(d) Specificity on time-series: ingress-04.csv



(e) MCC on time-series: ingress-05.csv

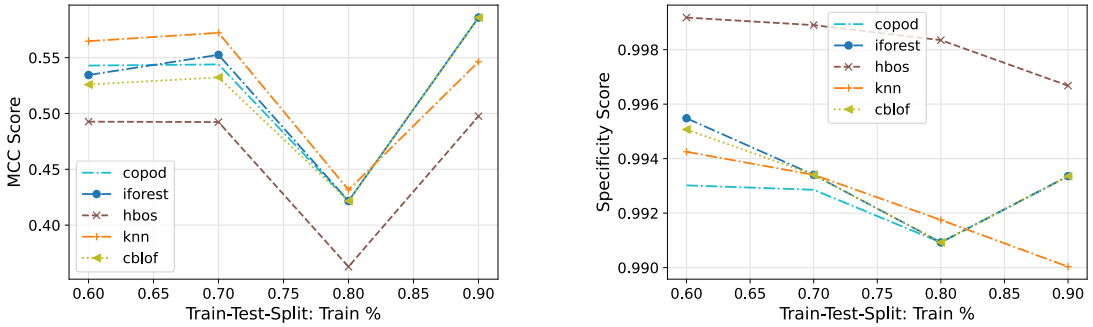


(f) Specificity on time-series: ingress-05.csv

Figure 6.15: MCC & specificity scores of all algorithms on dataset: data-ingress-rate

Dataset: ecommmerce-api-incoming-rps

Figure 6.16a shows the MCC scores on the time-series api-01.csv. All algorithms behave the same on the dataset, the MCC score drops sharply for all at a *train-test-split* of 0.8 compared to a *train-test-split* of 0.7, and exceeds all previous values at *train-test-split* of 0.9. This behaviour can only be explained by the nature of the dataset and the *train-test-split*. However, the dataset shows no abnormalities except for the shift of one anomalous datapoint section from the test dataset to the training dataset when changing from 0.7 to 0.8. The Specificity score in figure 6.16b offers no relevant insight, interestingly the number of false-positives increases and the specificity score decreases, except for COPOD and CBLOF, which have the highest MCC scores at 0.9 probably for this reason.



(a) MCC on time-series: api-01.csv

(b) Specificity on time-series: api-01.csv

Figure 6.16: MCC & specificity of all algorithms on dataset: ecommerce-api-incoming-rps

6.2.2 Outlier detection on anonymized commercial datasets

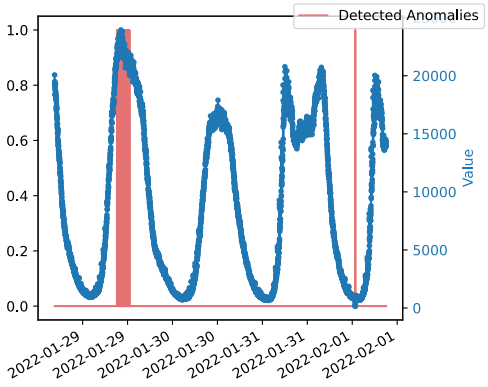
The anonymized commercial datasets cover a longer period than the datasets before, the period from 01.08.2021 to 01.02.2022. This also means that a typical *train-test-split* of 0.8 (80% training data, 20% test data) only represents the concrete use case to a limited extent, since a period of 2 hours is the standard for the "detection" mode. A *train-test-split* of 0.98 makes more sense, since this includes only about 3.5 days of data instead of about 37 days. Every algorithm was evaluated with a *train-test-split* of 0.8 and 0.98, and a contamination of 0.5% and 0.1%, so for every dataset and every algorithm, 4 evaluations and graphs were created. For the sake of clarity, only selected algorithms that show particularly interesting behaviour are selected for each time-series. All time-series are not labelled, the anomalies are recognisable due to the characteristics

of the time-series and the existing context and are evaluated according to this scheme.

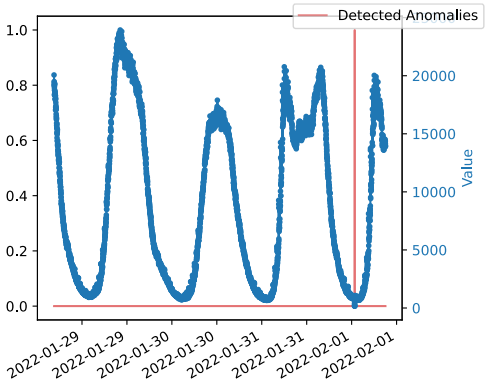
Time-series: Application 4

Looking at the *train-test-split* of 0.98, no directly visible anomalies are in the examined area, but an individual anomaly in the sub-context, at 01.02.2022 datapoints are untypically far down in relation to the other points in this daily seasonality. This anomaly was detected exclusively by the KNN algorithm (with both contamination rates of 0.5% in figure 6.17a and 0.1% in figure 6.17a, but with the trade-off that with 0.5, a lot of other points were detected as anomalies. The other points are only a common upward swing and not an outlier, making the detection false-positive. All algorithms with a contamination rate of 0.1% did not detect any anomalies in the dataset. Interestingly, all algorithms except HBOS produced false positives in the first peak of the time-series on 29.01.2022 with a contamination rate of 0.5. With a contamination rate of 0.1%, no algorithm except KNN detected an anomaly, so it can be clearly stated that for this time-series and this section KNN performed best.

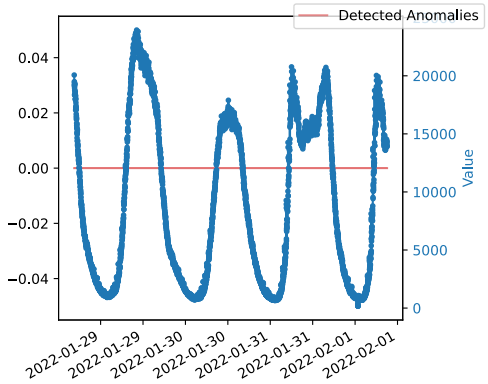
The results of evaluating the algorithms with a *train-test-split* of 0.8 give the same results, but with a contamination rate of 0.5 more false positives for all algorithms except HBOS.



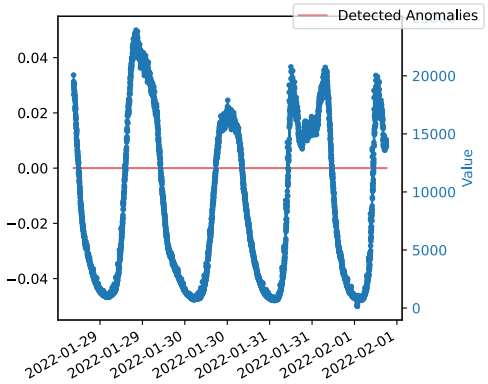
(a) Detected anomalies of KNN (Cont: 0.5%) on time-series: reg1-app4.csv



(b) Detected anomalies of KNN (Cont: 0.1%) on time-series: reg1-app4.csv



(c) Detected anomalies of HBOS (Cont: 0.5%) on time-series: reg1-app4.csv



(d) Detected anomalies of HBOS (Cont: 0.5%) on time-series: reg1-app4.csv

Figure 6.17: Detected Anomalies on time-series: reg1-app4.csv with *train-test-split* of 0.98

Time-series: Application 8

Application 8 is present in both regions, and as explained earlier in this chapter, the behaviour of the two anomalies does not correlate except for a change in the amplitude of the dataset. When analysing the time-series of region 1 app 8, it is noticeable that all algorithms with both contamination rates (0.5% and 0.1%) detect no anomalies (and thus no false positives). Since no algorithm is better or worse suited for this time-series and this *train-test-split*, a *train-test-split* of 0.3 is used for comparison. This is not a common distribution in machine learning, but it is not problematic because the training

domain contains enough data points, and only the immediate beginning of the test data (which also contains anomalies) is considered for the evaluation. Figure 6.18 shows the performance of KNN on the time-series with a *train-test-split*, a contamination rate of 0.0375% (which is the contamination rate of 0.1% adjusted to the smaller interval), and all algorithms detect the visible anomalous datapoints, and no false-positive datapoints. However, using a higher contamination rate of 0.1%, COPOD and HBOS detect two additional false-positives, while iForest and CBLOF detect 6 false positives, and KNN detects a large number of false positives (can be seen in figure 6.19). As a result, all algorithms perform well on this dataset, but the KNN, iForest, and CBLOF algorithms are much more sensitive to changes in the contamination rate, and this rate is crucial for a low number of false positives.

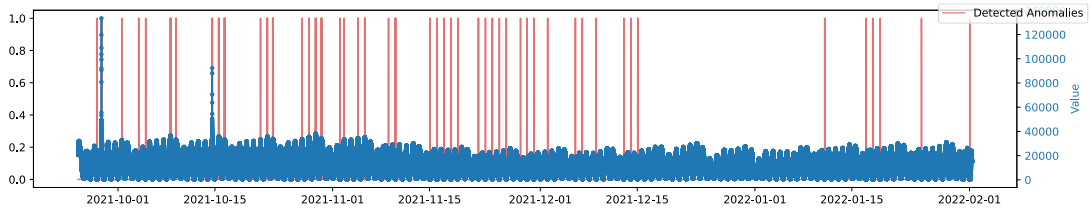


Figure 6.18: Detected anomalies of KNN (Cont: 0.1%) on time-series: reg1-app8.csv

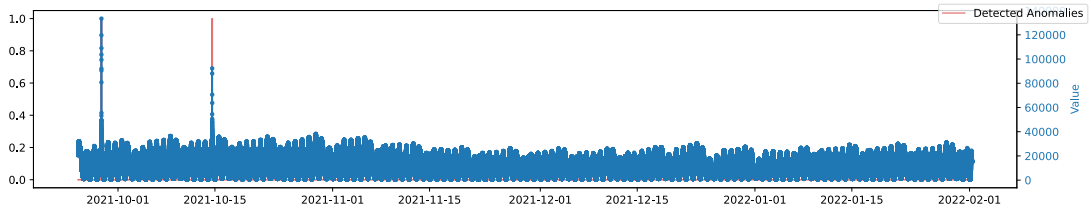
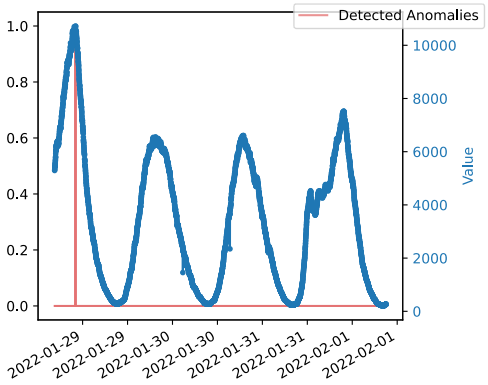
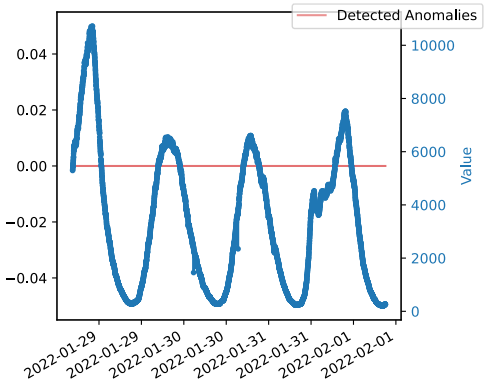


Figure 6.19: Detected anomalies of KNN (Cont: 0.0375%) on time-series: reg1-app8.csv

The time-series of application 8 in region 2 also does not contain any anomalies that algorithms should recognise with a *train-test-split* of 0.98. The first peak on 29.01.2022 can be considered an anomaly at first glance, as it is minimally higher than the other peaks of the previous month, but in the overall context of the time-series it is not an anomaly point. All algorithms do not detect any outliers with a contamination rate of 0.1% (figure 6.20b, but with a contamination rate of 0.5%, KNN detects the datapoint of the first peak (in figure 6.20a, which is again a sign that the algorithm reacts very finely to the contamination rate.



(a) Detected anomalies of KNN (Cont: 0.5%) on time-series: reg2-app8.csv

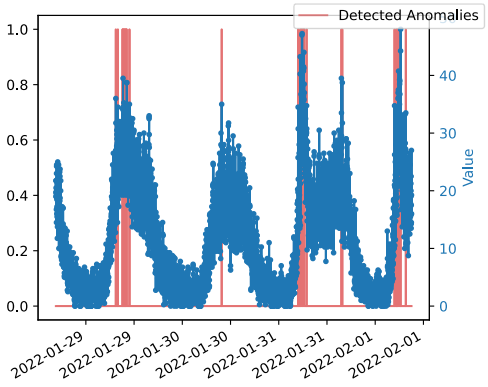


(b) Detected anomalies of KNN (Cont: 0.1%) on time-series: reg2-app8.csv

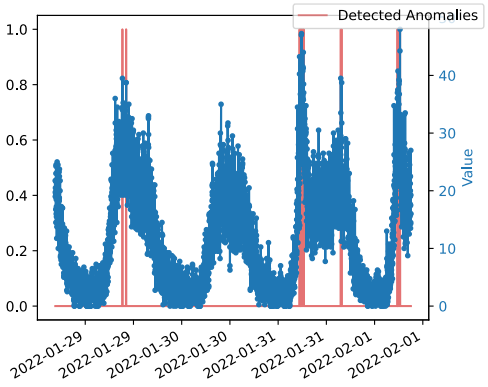
Figure 6.20: Detected Anomalies on time-series: reg2-app8.csv with *train-test-split* of 0.98

Time-series: Application 17

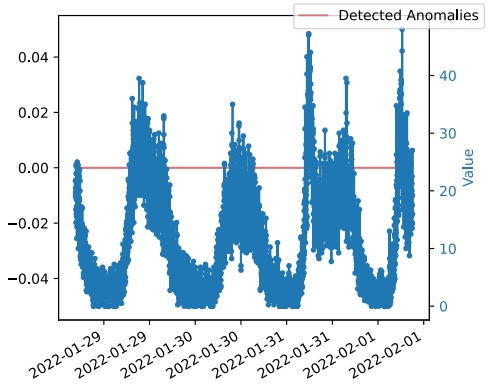
Application 17 is again available in both regions. In the region, the time-series shows interesting behaviour. Due to the steady increase in amplitude, there are no recognisable anomalies in the dataset, and the individual peaks can be interpreted with normal behaviour. If the algorithms are trained with a very small contamination rate, such as 0.0001%, then no algorithm will detect anomalies, and thus no false positives (see figure 6.21c and 6.21d). Nevertheless, to compare the algorithms, they were trained with a contamination rate of 0.1%, and the algorithms show different behaviour. HBOS detected only the isolated peaks in figure 6.21b (as did KNN), while iForest, CBLOF and COPOD (in figure 6.21a) also marked other points around these peaks, and also relatively smaller deflections.



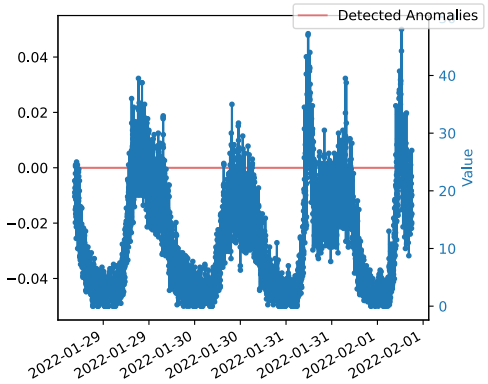
(a) Detected anomalies of COPOD (Cont: 0.1%) on time-series: reg1-app17.csv



(b) Detected anomalies of HBOS (Cont: 0.1%) on time-series: reg1-app17.csv



(c) Detected anomalies of COPOD (Cont: 0.0001%) on time-series: reg1-app17.csv

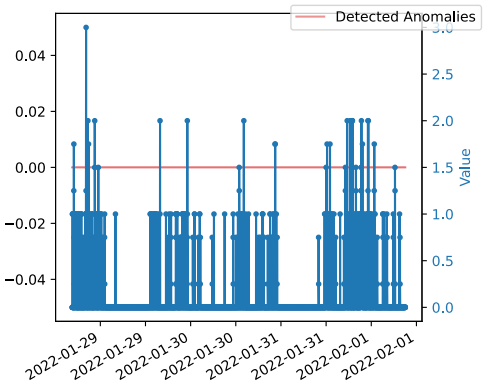


(d) Detected anomalies of HBOS (Cont: 0.0001%) on time-series: reg1-app17.csv

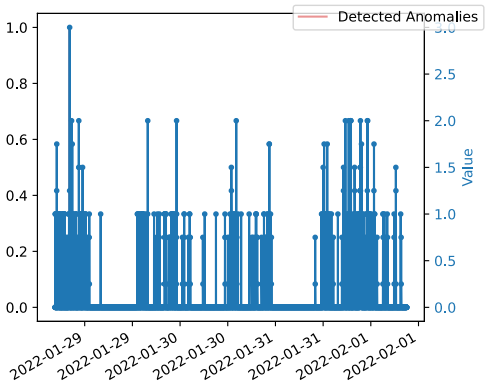
Figure 6.21: Detected Anomalies on time-series: reg1-app17.csv with *train-test-split* of 0.98

Application 17 in region 2 has a different character, but also, like in region 1 no anomalies in the test data. Overall, however, the dataset already contains anomalies, so that a contamination rate of 0.1% allows meaningful results to be obtained about the algorithms. KNN and HBOS correctly detect no anomalies here (figure 6.22a), while COPOD, CBLOF and iForest falsely detect the first peak with the value 3.0 (figure 6.22b. The reason for this behaviour may be the sensitivity to the contamination rate of COPOD, iForest, and CBLOF, which we check by setting the contamination rate to 0.5%. Our assumption was confirmed, as figure 6.22d, CBLOF immediately identifies all further peaks as anomalies, while KNN is for this time-series more robust to changes

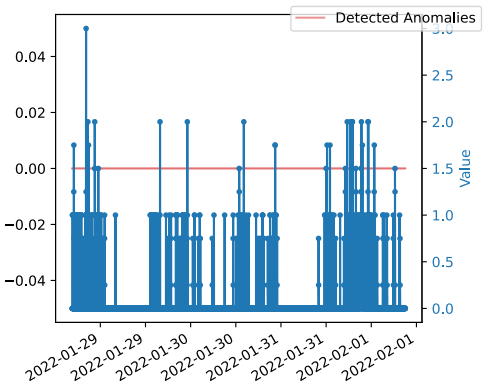
in the contamination rate.



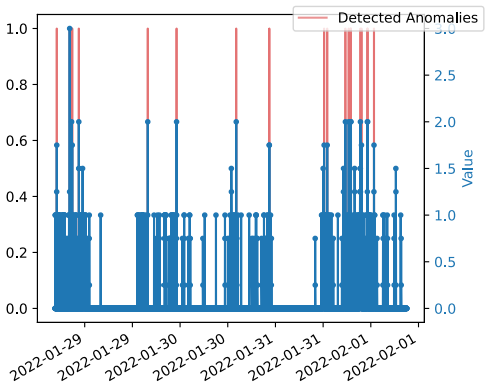
(a) Detected anomalies of KNN (Cont: 0.1%) on time-series: reg2-app17.csv



(b) Detected anomalies of CBLOF (Cont: 0.1%) on time-series: reg2-app17.csv



(c) Detected anomalies of KNN (Cont: 0.5%) on time-series: reg2-app17.csv



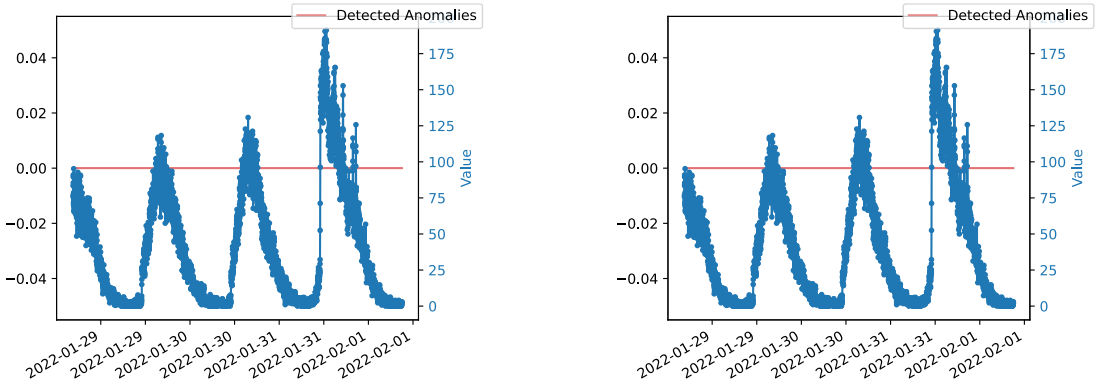
(d) Detected anomalies of CBLOF (Cont: 0.5%) on time-series: reg2-app17.csv

Figure 6.22: Detected Anomalies on time-series: reg2-app17.csv with *train-test-split* of 0.98

Time-series: Application 18

Application 18 in Region 2 is a time-series with strongly varying amplitude in seasonality with some anomaly points at the beginning of the dataset. Towards the end of the dataset, no anomalies are present, so the algorithms should not produce outliers and therefore, no false-positives. As the figures 6.23a and 6.23b exemplary show, all algorithms detect no anomalies and therefore fulfill the expectations of no

false-positives.



(a) Detected anomalies of iForest (Cont: 0.1%) on time-series: reg2-app18.csv

(b) Detected anomalies of COPOD (Cont: 0.5%) on time-series: reg2-app18.csv

Figure 6.23: Detected Anomalies on time-series: reg2-app18.csv with *train-test-split* of 0.98

6.2.3 Interesting optimisation through weakly supervised learning

The performance of the previous algorithms could be improved by changing the training phase.

Provided that the datasets are labelled, the training data can be changed so that the first feature of the time-series is the value, and the second feature is the label of whether a point is an anomaly or not (with the values 0 for normal point, 1 for anomaly). We call this weak-supervision since it can be used with unsupervised outlier detection algorithms without the need for code changes.

One problem that arises is that the test time-series must not contain fewer features than the normal one. Since adding the anomaly label to the testing time-series does not make sense, the feature label is also added to the testing time-series, but entirely as a value of 0 for each datapoint (the algorithm should not know in advance which points are anomalies).

For the algorithm COPOD on the time-series app1-01.csv from the dataset application-crash-rate-1, this method allows to reduce the number of false positives and increase the number of true negatives, which can be seen in the figures 6.24 and 6.25. This method needs further research to prove its usefulness and success.

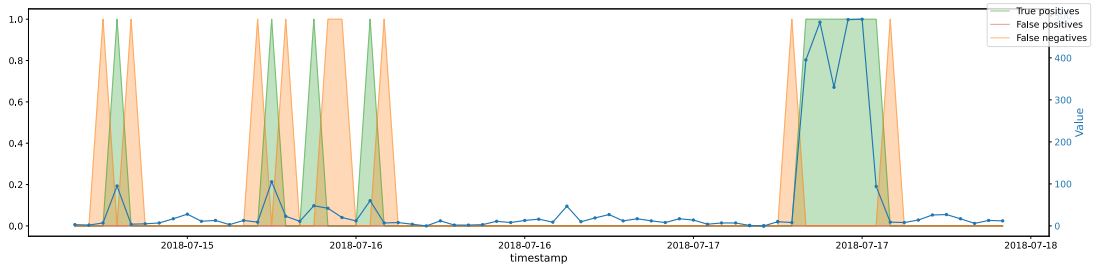


Figure 6.24: Detected anomalies using weak-supervised COPOD with *train-test-split* of 0.8

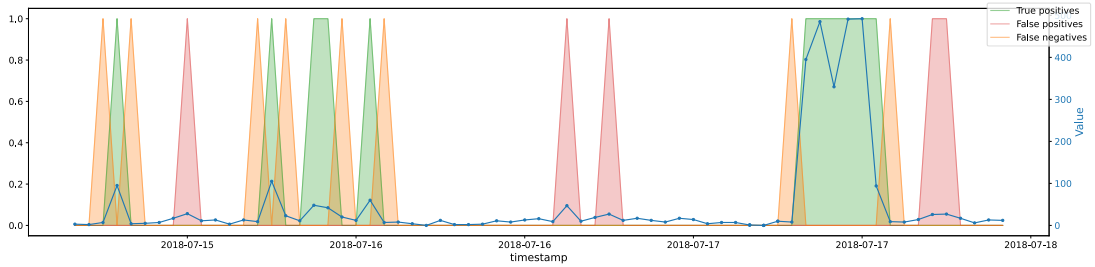


Figure 6.25: Detected anomalies using unsupervised COPOD with *train-test-split* of 0.8

6.3 Comparison of non-functional criteria

In this section, the non-functional criteria defined in chapter 4 are evaluated for the different anomaly detection algorithms. The implementation of all algorithms comes from the PyOD library [37], and every algorithm supports two functions, *fit()* and *predict()*, which are used through the evaluations and in the FADIP platform. The input parameters for the *fit()* (training) and *predict()* methods are *numpy* arrays with the shape (n_samples, n_features) or *pandas* dataframes, for time-series dataframes with timestamp and value as columns and rows for the single datapoints. Additionally, it has to be said that all algorithms presented here score very well when it comes to maintainability and community, as all are implemented in *PyOD*, and *PyOD* has an active community with many contributions [37].

iForest

The unsupervised outlier detection algorithm iForest was initially implemented in the *scikit-learn* library. *PyOD* wraps iForest of *scikit-learn* and adds more functionalities [1]. iForest (as implemented in *PyOD*) has many different input parameter, that can modify

the behaviour.

Configuration & setup is simple and can be completed in a short time when using either the implementation in *scikit-learn* or *PyOD*. A simple import of the library and the algorithm is ready for usage. There is no input parameter required to let the algorithm run. However, the contamination is a relevant input parameter that determines the ability to detect anomalies heavily. Other input parameters, like the maximum number of samples that the algorithm takes of the training set or the maximal amount of features to train the model, are also relevant.

In the iForest implementation of *PyOD*, it has 8 tuning options that allow tuning the performance of the algorithm in different dimensions, better model prediction performance, faster processing times for training and prediction. All tuning options (= input parameters) come with default values.

iForest is in the middle range in terms of user experience because the scores on the labeled open-source datasets show that iForest produces many false positives. On the anonymized commercial datasets, iForest is prone to produce false positives, especially on dense data points that differ from the rest of the time-series.

HBOS

HBOS, histogram-based outlier detection is again implemented in *PyOD*. The setup is again straightforward, the library needs to be imported, and there are no mandatory input parameters. However, contamination is an important parameter, it defines the threshold of the decision function. HBOS supports two working modes, a static number of bins as well as an automatic number of bins. [1]

HBOS supports 4 tuning parameters, the number of used bins, alpha as a regularizer for preventing overflow, tol as flexibility for dealing with samples that fall outside of the bins, and contamination as a threshold for the decision function. All input parameters have defined standard values, but the model prediction performance and the processing time can be influenced by the parameter choice. [1]

In the overall evaluation, HBOS proved to be one of the best algorithms in terms of correct classification (and therefore, user experience). Looking at the anonymized commercial datasets, HBOS also produces a low number of false positives, and together with the minor tuning options, it is therefore particularly suitable for productive use. HBOS stood out in particular for its fast execution time.

COPOD

COPOD, another outlier detection algorithm, implemented in *PyOD*, is just as simple in the setup. Also one import of the library is enough, and the algorithm is ready to start.

COPOD does not have a mandatory input parameter. Nevertheless, it is recommended that contamination is used as input parameter to define the threshold of the decision function. [1]

COPOD has the least tuning options. It only allows to specify the contamination rate of the training dataset to influence the performance of the predictions, and n_jobs as the input parameter to adjust the number of parallel jobs for fit and predict (can result in faster execution times). Both parameters come with default values. [1]

COPOD has an interesting performance in the overall evaluation. For some time-series the algorithm performs very well, while others produce poor results. This also applies to the anonymized commercial datasets, where there were more false positives than would be pleasant for the user experience. COPOD also offers hardly any additional possibilities to improve the performance of the algorithms using parameters.

KNN

The k-Nearest Neighbors Detector (KNN) is implemented in the *PyOD* library, interpreting the distance of a point to its k-th nearest neighbors as the outlier score. KNN is a very flexible, configurable outlier detection method. [1]

KNN allows many different input parameters to adjust the performance of the predictions. In total, 10 input parameters for the initialization of the algorithm are possible, some having a stronger influence on the prediction performance (like contamination, n_neighbors, and radius), and some having minor to no influence on prediction performance (like algorithm and n_jobs). [1]

The performance of KNN was also very dependent on the selected dataset. For highly varying datasets the performance was not so good, and the number of false positives was high. For periodic datasets the performance was partly excellent. Particularly on the anonymized commercial datasets, the algorithm was for some cases the only one to detect certain anomalies but was also more sensitive to the contamination rate for some datasets. Thus, no correct statement can be made about the user experience of anomaly detection with this algorithm. It should be noted, however, that KNN took a long time for training and prediction.

CBLOF

PyOD also provides the implementation of Clustering Based Local Outlier Factor, an anomaly detection algorithm that works by clustering and separating the clusters into small and large clusters. The setup is again convenient; an import of the library and no mandatory input parameter lead to a fast setup.

CBLOF has 8 input parameters as tuning options to improve the performance and the

execution time of the algorithm. Contamination is again used as the input parameter to define the threshold of the decision function. Also, the number of clusters, the base clustering algorithm, and the values for dividing into small and large clusters can be varied. All input parameters come with standard values.

The performance of CBLOF is also highly dependent on the datasets; for the labelled datasets, the algorithm was among the best for some and among the worst for others. In the anonymized commercial datasets, CBLOF detects many anomalies that turn out to be false positives, especially with poorly chosen contamination rates. It is also difficult to assess the user experience for anomaly detection for CBLOF.

6.3.1 Summary

In summary, it can be said that the performance of all algorithms depends very much on the datasets used. HBOS generally performed well on many datasets, followed by KNN. However, the other algorithms tend to produce more false positives in general, but again this depends heavily on the dataset used. HBOS was also the fastest algorithms in terms of execution time, and KNN the slowest algorithm. All this leads to the conclusion that in the productive use of anomaly detection, a prior evaluation of the algorithms on the time-series to be used must take place. Ideally, this evaluation takes place with labelled time-series, but usually this is not possible, in which case an evaluation with unlabelled ones will have to suffice.

7 Conclusion & Outlook

The main focus of this thesis is to develop a flexible anomaly detection platform that allows the evaluation of different anomaly detection algorithms on different open-source and anonymized car monitoring datasets. Furthermore, the platform should allow the usage of the evaluated algorithms in productive IT systems to detect anomalies in the time-series data from monitoring systems.

In chapter 2, we gave a basic understanding of anomalies and showed that there are three different types: individual outliers in the context of the complete time-series, individual outliers in a sub-context, and grouped outliers. Then, the application domain of car backend monitoring was defined, and we presented the challenges of monitoring a complex system stack and how the microservice approach of Thanos.io solved them.

Next, fair and measurable comparison criteria were defined, divided into functional and non-functional criteria. The functional criteria consisted of different formulas used to measure the direct performance of the algorithms on a dataset, and the MCC score and the specificity were presented as reliable comparison metrics. The F1 score was analysed and because of the easy possibilities to influence it with the help of the train-test-split, the MCC score was favoured. The non-functional criteria were defined as usability of the algorithms and tuning options, resulting in that algorithms with fewer false-positives are preferable for this criterion.

In chapter 5, the complex requirements that a production-ready system must fulfil are shown, and a flexible anomaly detection integration platform (FADIP) was presented. It was demonstrated how FADIP fulfills all presented requirements, and the platform, extendability and evaluation and detection capabilities were explained in detail. FADIP is a very flexible, extendable solution that allows to use multiple anomaly detection algorithms easily for evaluation and detection purposes.

Since the main focus of the monitoring data were unlabeled time-series from car monitoring and labeled open-source time-series datasets, different datasets (unlabeled and labeled) with multiple time-series were chosen in chapter 6. Also, all algorithms' performance was compared (by the platform) on all these datasets using the MCC score and specificity. The results were mixed, HBOS performed better than most of the algorithms, and for the other algorithms, no accurate assessment could be made for all datasets.

However, as all comparisons on the outlier detection algorithms show, the choice of

the algorithm highly depends on the different characteristics of the time-series, and the input parameter contamination is very relevant as a threshold for deciding between normal and anomalous datapoints. A prior evaluation is necessary for the productive use of the algorithms, which is easily possible with FADIP; without that, the application in productive IT infrastructure is questionable.

7.1 Outlook

The platform itself has even more potential if it is structured more modularly. At the moment, only PromQL data sources are supported. The conversion to a modular system, which defines a uniform time-series format for univariate and multivariate time-series as well as a transformation component, can be used as a solution to add additional datasources in a modular way. Now that a common interface has been defined for the algorithms, there is definitely further potential for improvement. The current interface limits the adaptation of some algorithms to specific parameters only; this limitation could be solved with a more flexible interface, thus also opening the way for other implementations of algorithms, mainly supervised and not only unsupervised or weakly-supervised algorithms. The current implementation scales for the used datasets well, but it might run into a problem for larger datasets (especially multivariate datasets), so adding automatic resampling of data or switching to another framework for transforming and processing the data might be a good improvement. Scalability can also be achieved by developing the application to cluster computing methods like Apache Spark. To further improve usability, one can also consider developing a web interface. This web interface could then provide various information about the state of the system, the detected anomalies, and statistics. Above all, this would make improving supervised algorithms easier to flag false-positive and false-negative anomalies in past time-series data points and use this feedback to retrain the models. Overall, it can be said that this platform has great potential to deploy anomaly detection in professional cloud infrastructure in a production-ready manner and provide tangible benefits for monitoring IT systems.

List of Figures

List of Tables

Bibliography

[1] *All Models - pyod 0.9.7 documentation*. https://pyod.readthedocs.io/en/latest/pyod.models.html. Accessed: 2022-02-10.

[2] F. Angiulli and C. Pizzuti. "Fast Outlier Detection in High Dimensional Spaces." In: *Principles of Data Mining and Knowledge Discovery*. Ed. by T. Elomaa, H. Mannila, and H. Toivonen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 15–27. ISBN: 978-3-540-45681-0.

[3] *Benefits of Using Docker*. https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ETS-help/GUID-F5BDACC7-6F0E-4EBB-9F62-E0046D8CCF1B.html. Accessed: 2022-01-20.

[4] H. Breuing, L. Heil, and B. Vierling. "IT Security for the Entire Automotive Ecosystem." In: *ATZelectronics worldwide* 14.7 (2019), pp. 60–63. DOI: 10.1007/s38314-019-0076-7.

[5] D. Chicco and G. Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation." In: *BMC Genomics* 21.1 (Jan. 2020), p. 6. ISSN: 1471-2164. DOI: 10.1186/s12864-019-6413-7.

[6] *Data model | Prometheus*. https://prometheus.io/docs/concepts/data_model/. Accessed: 2021-10-24.

[7] *Effects of Bin Width and Height in a Histogram - Wolfram Demonstrations Project*. https://demonstrations.wolfram.com/EffectsOfBinWidthAndHeightInAHistogram/. Accessed: 2022-02-14.

[8] *FADIP Platform*. https://github.com/mstapfner/fadip. Accessed: 2022-02-10.

[9] D. Fourure, M. U. Javaid, N. Posocco, and S. Tihon. *Anomaly Detection: How to Artificially Increase your F1-Score with a Biased Evaluation Protocol*. 2021. arXiv: 2106.16020 [cs.LG].

[10] M. Goldstein and A. Dengel. "Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm." In: *KI-2012: Poster and Demo Track. German Conference on Artificial Intelligence (KI-2012), 35th, September 24-27, Saarbrücken, Germany*. Ed. by S. Wölfl. Online, Sept. 2012, pp. 59–63.

[11] M. Goldstein and S. Uchida. "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data." In: *PLOS ONE* 11.4 (Apr. 2016), pp. 1–31. DOI: 10.1371/journal.pone.0152173.

[12] Z. He, X. Xu, and S. Deng. "Discovering cluster-based local outliers." In: *Pattern Recognition Letters* 24.9 (2003), pp. 1641–1650. ISSN: 0167-8655. DOI: https://doi.org/10.1016/S0167-8655(03)00003-5.

[13] J. Hernantes, G. Gallardo, and N. Serrano. "IT Infrastructure-Monitoring Tools." In: *IEEE Software* 32.4 (2015), pp. 88–93. DOI: 10.1109/MS.2015.96.

[14] M. Hung. *Leading the iot, gartner insights on how to lead in a connected world.* 2017.

[15] *Isolation Forest | Anomaly Detection with Isolation Forest*. https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/. Accessed: 2022-02-14.

[16] *IT Infrastructure Monitoring Tools Reviews 2021 | Gartner Peer Insights*. https://www.gartner.com/reviews/market/it-infrastructure-monitoring-tools. Accessed: 2021-10-23.

[17] N. Jiang and L. Gruenwald. "Research Issues in Data Stream Association Rule Mining." In: *SIGMOD Rec.* 35.1 (Mar. 2006), pp. 14–19. ISSN: 0163-5808. DOI: 10.1145/1121995.1121998.

[18] T. Kessler and C. Buck. "How Digitization Affects Mobility and the Business Models of Automotive OEMs." In: *Phantom Ex Machina: Digital Disruption's Role in Business Model Transformation*. Ed. by A. Khare, B. Stewart, and R. Schatz. Cham: Springer International Publishing, 2017, pp. 107–118. ISBN: 978-3-319-44468-0. DOI: 10.1007/978-3-319-44468-0_7.

[19] Z. Lan, Z. Zheng, and Y. Li. "Toward Automated Anomaly Identification in Large-Scale Systems." In: *IEEE Transactions on Parallel and Distributed Systems* 21.2 (2010), pp. 174–187. DOI: 10.1109/TPDS.2009.52.

[20] A. Lavin and S. Ahmad. "Evaluating Real-Time Anomaly Detection Algorithms – The Numenta Anomaly Benchmark." In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)* (Dec. 2015). DOI: 10.1109/icmla.2015.141.

[21] Z. Li, Y. Zhao, N. Botta, C. Ionescu, and X. Hu. "COPOD: Copula-Based Outlier Detection." In: *2020 IEEE International Conference on Data Mining (ICDM)* (Nov. 2020). DOI: 10.1109/icdm50108.2020.00135.

[22] F. T. Liu, K. M. Ting, and Z.-H. Zhou. "Isolation-Based Anomaly Detection." In: *ACM Trans. Knowl. Discov. Data* 6.1 (Mar. 2012). ISSN: 1556-4681. DOI: 10.1145/2133360.2133363.

[23] *Matthews Correlation Coefficient is The Best Classification Metric You've Never Heard Of*. https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3bf50a2f3e9a. Accessed: 2021-02-01.

[24] *Microsoft Cloud Monitoring Dataset*. https://github.com/microsoft/cloud-monitoring-dataset. Accessed: 2021-01-06.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. 2nd. The MIT Press, 2018. ISBN: 0262039400.

[26] *Prognose zum Umsatz mit Enterprise-Software** weltweit von 2009 bis 2022*. https://de.statista.com/statistik/daten/studie/191804/umfrage/weltweiter-umsatz-mit-enterprise-software-seit-2009/. Accessed: 2021-10-21.

[27] *Revisiting PromCon 2018 Panel On Prometheus Long-Term Storage*. https://dev.to/mhausenblas/revisiting-promcon-2018-panel-on-prometheus-long-term-storage-5f1p. Accessed: 2022-02-05.

[28] S. Sadik and L. Gruenwald. "Research Issues in Outlier Detection for Data Streams." In: 15.1 (Mar. 2014), pp. 33–40. ISSN: 1931-0145. DOI: 10.1145/2594473.2594479.

[29] T. Smith and E. Frank. "Introducing Machine Learning Concepts with WEKA." In: *Methods in molecular biology (Clifton, N.J.)* 1418 (Mar. 2016), pp. 353–378. DOI: 10.1007/978-1-4939-3578-9_17.

[30] *Software - Worldwide | Market Forecast*. https://www.statista.com/outlook/tmo/software/worldwide. Accessed: 2021-10-21.

[31] *Thanos - Highly available Prometheus setup with long term storage capabilities*. https://thanos.io/tip/components/sidecar.md/. Accessed: 2022-02-05.

[32] *Thanos - Highly available Prometheus setup with long term storage capabilities*. https://thanos.io/tip/components/store.md/. Accessed: 2022-02-05.

[33] *Thanos - Highly available Prometheus setup with long term storage capabilities*. https://thanos.io/tip/components/query.md/. Accessed: 2022-02-05.

[34] J. Turnbull. *Monitoring with Prometheus*. Turnbull Press, 2018.

[35] *What Is a Good F1 Score? - Inside GetYourGuid*. https://inside.getyourguide.com/blog/2020/9/30/what-makes-a-good-f1-score. Accessed: 2021-02-01.

[36] *What is Amazon S3? - Amazon Simple Storage Service*. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. Accessed: 2022-02-05.

[37] *yzhao062/pyod: (JMLR' 19) A Python Toolbox for Scalable Outlier Detection (Anomaly Detection)*. https://github.com/yzhao062/pyod. Accessed: 2021-02-08.

[38] *Zeitreihen-basiertes Monitoring mit Prometheus*. https://www.linux-magazin.de/ausgaben/2017/06/prometheus/. Accessed: 2021-04-06.