University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Computing the Trustworthiness Level of Unsupervised AI-based Intrusion Detection Systems

Mauro Doerig
Appenzell, Switzerland
Student ID: 18-731-919

Supervisor: Dr. Alberto Huertas Celdran, Muriel Franco
Date of Submission: September 14, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Systems utilizing artificial intelligence (AI) are becoming more and more useful in supporting human decision-making tasks. Unsupervised AI algorithms play a significant role in the detection of intrusions and cyberattacks on devices with limited resources. However, current solutions focus on achieving the best detection performance, missing the importance of quantifying the trustworthiness level of the trained models and their predictions. This work focuses on computing the level of trustworthiness for unsupervised anomaly detection models. In this work a taxonomy with four different pillars of trust and associated metrics is proposed. Further, an algorithm has been developed which takes unsupervised anomaly detection models together with the underlying training, test and outlier data sets as inputs to compute an overall trust score. This algorithm has further on been embedded in a web application which is available to model developers and serves as a global solution to evaluate unsupervised anomaly detection models towards trustworthiness. Lastly, an in-depth analysis on different models has been conducted in order to evaluate the proposed algorithm and to point out strengths and limitations.

Zusammenfassung

Systeme, die künstliche Intelligenz (KI) nutzen, werden immer häufiger zur menschlichen Entscheidungsfindung herangezogen. Unüberwachte KI-Algorithmen spielen eine wichtige Rolle bei der Erkennung von Cyberangriffen auf Geräten mit begrenzten Ressourcen. Aktuelle Lösungen konzentrieren sich jedoch darauf, die beste Erkennungsleistung zu erzielen und vernachlässigen dabei die Wichtigkeit der Vertrauenswürdigkeit solcher Modelle und ihrer Vorhersagen. Diese Arbeit konzentriert sich auf die Berechnung des Vertrauenswürdigkeitsgrades für unüberwachte Anomalieerkennungsmodelle. In dieser Arbeit wird eine Taxonomie mit vier verschiedenen Säulen des Vertrauens und zugehörigen Metriken vorgeschlagen. Darüber hinaus wurde ein Algorithmus entwickelt, der Anomalieerkennungsmodelle zusammen mit den zugrundeliegenden Trainings-, Test- und Ausreisserdatensätzen als Eingaben verwendet, um einen allgemeinen Vertrauenswert zu berechnen. Dieser Algorithmus wurde in eine Webanwendung eingebettet, die Modellentwicklern zur Verfügung steht und als globale Lösung zur Bewertung von unüberwachten Anomalieerkennungsmodellen hinsichtlich ihrer Vertrauenswürdigkeit dient. Abschliessend wurde eine eingehende Analyse verschiedener Modelle durchgeführt, um den vorgeschlagenen Algorithmus zu evaluieren sowie dessen Stärken und Grenzen aufzuzeigen.

Acknowledgments

First and foremost, I want to express my gratitude to the Communication Systems Research Group at UZH for their assistance and support during the project. I would like to express special thanks to Dr. Alberto Huertas and Prof. Dr. Burkhard Stiller for enabling me to work on this project and consistently supporting me in both words and acts.

A special thanks also goes out to Jan Bauer, who helped me dive into the subject and the rest of his team that laid the foundation for this work.

After half a year of dedicated work, I'm happy how far we came with the project, and I'm proud of what I, with all the support around me, have accomplished. I appreciate the unwavering support that my family and partner have given me throughout this challenging period.

Contents

Abstract **i**

Acknowledgments **v**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 2

1.3 Thesis Outline 2

2 Related Work **3**

2.1 Related Work in Supervised ML Environment 3

2.2 Related Work in Unsupervised ML Environment 4

2.2.1 Literature 4

3 Computing Trustworthiness for Unsupervised Anomaly Detection **7**

3.1 Trustworthiness in Machine Learning 7

3.2 Trustworthy Anomaly Detection Introduction 7

3.2.1 Difference between Novelty Detection and Outlier Detection 8

3.3 Pillars of Trust 8

3.4 Fairness 8

3.4.1 Underfitting 8

3.4.2 Overfitting 9

3.4.3 Statistical Parity Difference 10

3.4.4 Disparate Impact . . . 11
3.4.5 Fairness Overview . . . 13
3.5 Explainability . . . 14
3.5.1 Feature Correlation . . . 14
3.5.2 Model Size . . . 15
3.5.3 Permutation Feature Importance Score . . . 16
3.5.4 Explainability Overview . . . 17
3.6 Robustness . . . 17
3.6.1 CLEVER Score . . . 18
3.6.2 Robustness Overview . . . 19
3.7 Methodology . . . 19
3.7.1 Normalization . . . 20
3.7.2 Regularization . . . 20
3.7.3 Missing Data . . . 21
3.7.4 Train Test Split . . . 21
3.7.5 Factsheet Completeness . . . 22
3.7.6 Methodology Overview . . . 22

4 Webapp **23**

4.1 Trusted-AI Webapp . . . 23
4.2 Algorithm Design . . . 23
4.3 Webapp Design . . . 28
4.4 Extension of Webapp for Unsupervised Anomaly Detection . . . 29
4.4.1 Scenarios Page . . . 29
4.4.2 Upload Page . . . 32
4.4.3 Analyze Page . . . 34
4.4.4 Compare Page . . . 38

5 Evaluation **41**

5.1 Scenario: IT Security - IoT Data Anomaly Detection 41

5.2 Solutions: IT Security - IoT Data Anomaly Detection 41

5.3 Model Analysis 42

5.3.1 Isolation Forest 45

5.3.2 Local Outlier Factor 47

5.3.3 Learning One-Class Support Vector Machine 48

6 Discussion **51**

6.1 Contribution 51

6.2 Limitations 52

7 Summary and Conclusions **53**

7.1 Future Work 54

7.1.1 Improvement of Computation Speed 54

7.1.2 Additional Robustness Metrics 54

7.1.3 Webapp Usability 54

7.1.4 Extension for Other Unsupervised Scenarios or Semi-Supervised Scenarios 54

List of Figures **56**

List of Tables **58**

Chapter 1

Introduction

1.1 Motivation

In the twenty-first century, artificial intelligence (AI) has made incredible advancements. Face recognition, natural language processing, audio recognition, text production, language translation, medicine research, and other technologies that use artificial intelligence are examples of its accomplishments [1]. As capabilities grew, AI became also more relevant as a tool to support human decision-making. In these scenarios it became obvious that we do not only rely on the performance of such models but also heavily on its trustworthiness. For instance, it was discovered that a AI-based software used by judges to score pretrial recidivism was racially prejudiced [19]. Similar discrimination against women was perpetrated by a model used to screen job applicants at a large technology company [2]. Examples like these had a far-reaching impact and led to higher awareness and thus to intensive research in the area of trustworthy AI.

Research has identified different pillars of trustworthiness to structure and further specify the term. Many different pillars of trustworthiness in AI are described in the literature of which fairness, explainability, robustness and methodology have crystallized out of it [11] [9] [12]. Most of this work has been done for supervised machine learning which is defined by its use of labeled data sets to train classification or prediction algorithms. Several metrics have been proposed to calculate and quantify trustworthiness and most of them rely on labels for its calculation. Metrics such as statistical parity, equal opportunity, average Odds, and other fairness measures are all included in IBM's AI Fairness 360 toolbox [11] and can be used to identify bias in machine learning models and datasets. The entire lifecycle of AI-based systems is covered by algorithms that can reduce bias during the pre-processing, in-processing, and post-processing stages. Note that the given example metrics rely on labeled data sets for their computation.

Difficulties arise when we want to compute metrics related to trustworthiness for unsupervised machine learning models where no labels are given. There is a whole new approach to it. The motivation of this thesis is to follow this new approach and develop an algorithm which computes the pillars of fairness for unsupervised machine learning models with existing and new metrics at hand.

1.2 Description of Work

Anomaly detection has several practical uses, including the detection of cyber attacks and bank fraud. Many anomaly detection models have been developed during the last ten years, which has greatly advanced the ability to accurately detect a variety of anomalies. Even though, the performance of detecting anomalies has substantially improved with enhanced anomaly detection approaches, there are still major hazards when using models in sensitive applications for automatic decision-making, such as financial or healthcare systems [23]. In the context of anomaly detection, trustworthy algorithms should adhere to common societal norms in addition to being effective in detecting anomalies.

This Bachelor thesis will design, develop, integrate, and evaluate an algorithm that can quantify the degree of trustworthiness of unsupervised machine learning and deep learning models focusing on anomaly detection.

1.3 Thesis Outline

The thesis is structured into seven chapters.

Chapter 2 goes more into detail about existing theoretical and practical work which has been done in the research area of trustworthy AI. This chapter has a research nature and paves the way to the practical part of this thesis: implementing metrics to compute trustworthiness for unsupervised models.

Chapter 3 takes a closer look at the proposed pillars and its metrics. Fairness, explainability, robustness, and methodology are described. The key metrics for analysing a model towards trustworthiness are discussed for each pillar. Also the approach of adjusting metrics proposed for supervised models into metrics for unsupervised models is given.

In Chapter 4 the described pillars and metrics from Chapter 3 are taken and built into an algorithm which computes the final trust score of a model. This new algorithm is named Trusted-Anomaly-Detection. A python-based webapp is then extended by this algorithm and allows in addition to supervised models also unsupervised anomaly detection models to be evaluated.

Chapter 5 contains the evaluation of the proposed algorithm. Different unsupervised anomaly detection models are taken at hand and fed into Trusted-Anomaly-Detection.

Chapter 6 addresses the contributions of this bachelor thesis and further on points out limitations of the conducted work.

Finally, Chapter 7 provides a summary and conclusion on the bachelor thesis. Also, limitations and future work are discussed.

Chapter 2

Related Work

In this part of the bachelor thesis, an in-depth literature review was conducted focused first on supervised metrics and thereafter on unsupervised metrics relating to trustworthiness. The literature research became increasingly more specific to unsupervised anomaly detection, with first a global search for all possible existing metrics and then for each pillar individually. The related work found is described in the following section.

There has been no tool been found which is able to quantify the trustworthiness of unsupervised anomaly detection models. However, this bachelor thesis is strongly oriented towards an already existing tool, which is able to do exactly this for supervised models. The work of Jan Bauer, Joel Leupp and Melike Demirci [12] on an algorithm and webapp named ”Trusted-AI” for calculating the trustworthiness of supervised models serves as the basis for this bachelor thesis.

In addition to Trusted-AI, many other literature sources have been consulted, which have led to a great deal of work on the individual pillars of trust which are separated into supervised and unsupervised machine learning (ML) categories in the following part.

2.1 Related Work in Supervised ML Environment

Fairness metrics such as statistical parity, equal opportunity, average odds, and other metrics for supervised ML models are all included in IBM's AI Fairness 360 toolbox [11] and can be used to identify bias in machine learning models and datasets. The entire lifetime is covered by algorithms that can reduce bias during the pre-processing, in-processing, and post-processing stages.

A python library called Fairlearn gives creators of AI systems the ability to evaluate the fairness of their systems and address any concerns with reported unfairness. Fairlearn includes metrics for model evaluation as well as mitigation algorithms. This repository includes Jupyter notebooks with usagesvasa examples for Fairlearn in addition to the raw code [6].

Numerous explainability algorithms and methods are included in IBM's AI Explainability 360 toolbox [10]. The algorithms aim to develop explanations for ML models or aid in improving

the accuracy of models like decision trees that are already explainable. It is not about rating the degree of global model explainability, but rather about evaluating explanation methods for various models. Additionally, many Python libraries have local explanatory methods like LIME or SHAPE available. All of those toolkits are designed to explain model decisions, not to categorize models according to how easily they can be understood.

The IBM-provided Adversarial Robustness Toolbox (ART) [9] is an open-source Python toolbox that contains a wide range of adversarial robustness methods for supervised machine learning models. It includes adversarial attack and defense implementations, mechanisms for detecting attacks in real time, a way to detect poisoning and robustness metrics [7]. They offer a library that may be downloaded, imported and utilized while writing Python code. Additionally, ART employs a variety of attack detection and defense strategies that are not the primary focus of the reliable AI algorithm.

2.2 Related Work in Unsupervised ML Environment

2.2.1 Literature

Trustworthiness for unsupervised anomaly detection is described in different pillars and properties. Trustworthy anomaly detection models should be performant, interpretable, fair, robust and privacy-preserving [23].

Performance: All anomaly detection models must, at a minimum, be capable of accurately detecting anomalies. Aiming to decrease the rate of incorrectly classifying normal samples as anomalies, the rate of missing anomalies, or both.

Interpretability: It is important to understand why models predict some data points as anomalies, since for users and providers of models miss-predictions can be a threat to both. For instance, both the account holder and the bank would like an explanation, such as what actions caused a suspension, if a bank account were to be automatically suspended by algorithms built to suspected fraudulent activity.

Fairness: There should be no algorithmic bias against specific groups in anomaly detection algorithms, as this is a fundamental ethical need from the general public.

Robustness: The anomaly detection model must have consistent outputs while confronted by attacking samples in order be robust and save. When models are used in real-world settings, such as identifying outliers in roadways for intelligent transportation systems, model robustness is essential for maintaining the model's reliability.

Privacy-Preservation: Sensitive user data must be properly protected, as mandated by laws and regulations. Anomaly detection should be able to secure the privacy of data in three different ways: training data used to train a model, the model itself, and the model's predictions. In the entire process of training and deploying an anomaly detection model, private user information shouldn't be disclosed.

Apart from advances in classifying and describing different aspects of trustworthiness in anomaly detection, scientific work in trustworthy unsupervised anomaly detection has mainly focused on proposing more trustworthy models by improving the above stated properties in standard models. As an example, the work of Hongjing Zhang and Ian Davidson proposes a new architecture and model for fair anomaly detection called “Deep Fair SVDD”. This model is trained such that the relationships between the sensitive attributes and the learned representations are decorrelated which improves the fairness of the model [25]. As another example, Minh-Nghia Nguyen and Ngo Anh Vien proposed an Autoencoder-Based One-Class Support Vector Machine called “AE-1SVM” which allow to study decision making for anomaly detection and thus makes the model more interpretable [17].

However, it seems that even after intensive literature research, no global solution can be found, being able to measure pillars of trust in terms of applicable metrics.

Chapter 3

Computing Trustworthiness for Unsupervised Anomaly Detection

3.1 Trustworthiness in Machine Learning

Statistical models developed using deep learning and machine learning techniques are increasingly being used as core components of products and services. It takes more faith to use these algorithms in high-stakes situations like credit applications, court rulings and medical advice [3]. But according to a survey of corporate executives, they find it very difficult to put their trust in AI systems. As humans we rely a lot on automated systems. As an illustration, we rely on an airplane's autopilot and accept that it will fly safely. According to IBM, the four main pillars of fairness, explainability, robustness and methodology are what people trust in AI. A transparent AI life cycle is required in order to guarantee trust in outcomes based on AI. The key design decisions need to be documented at every stage of the life cycle.

3.2 Trustworthy Anomaly Detection Introduction

The goal of anomaly detection is to identify instances that differ from typical ones. Anomaly detection has drawn a lot of interest recently as a classic machine learning topic with numerous applications. The performance of detecting anomalies has substantially improved with enhanced anomaly detection approaches, however there are still major hazards when using models in sensitive applications for automatic decision-making, such as financial or healthcare systems. The detection of bank fraud, online trolls and malicious insiders are just a few of the anomaly detection tasks involving humans. Because every individual has the potential to be a victim of algorithms due to intentional or inadvertent misuse, it is only natural to question whether we should trust algorithms to identify people as anomalies.

Designing reliable AI algorithms is the fundamental goal in order to meet social expectations for increasing AI capability and societal benefits without endangering anyone or posing a threat to them.

Anomaly detection is typically carried out in unsupervised and semi-supervised contexts due to the limited amount of anomalies in the training data set. According to the underlying nature of data distributions in real-world scenarios, the unsupervised anomaly detection setup often assumes unlabeled data in the training set with rare anomalies [23].

3.2.1 Difference between Novelty Detection and Outlier Detection

In novelty detection, you have a data set that contains only good data and you are trying to check if new observations are similar to the the good data. In other words, our goal is to check if new observations are outliers. In outlier detection, the data set may already have outliers and your goal is to identify them. Both novelty detection and outlier detection are used to detect anomalies. Outlier detection is an unsupervised anomaly detection algorithm. The term "outlier detection" and "anomaly detection" is used interchangeably in the following sections.

3.3 Pillars of Trust

The following sections describe the pillars of trust that emerged from the literature review. In addition, selected metrics for each pillar are listed and explained. At the end of each section, an overview of the pillar and metrics is provided.

3.4 Fairness

As machine learning dominated more application domains in recent years, fairness became more and more crucial. When a system or decision treats certain protected elements impartially or neutrally, it is typically referred to as fair. However, the precise meaning of fairness relies on the context and the defining subject.

Legislative regulations exist to protect people against prejudice based on delicate or protected characteristics like religion, ethnicity or demographics. For instance, the American Fair Housing Act of 1968 forbids housing discrimination based on racial, ethnic, national, religious, sex, family and disability reasons [24].

In the following different metrics are taken at hand to evaluate fairness of the model and its underlying data. It is also discussed how metrics which were originally proposed for supervised solutions are adjusted to evaluate unsupervised anomaly detection models.

3.4.1 Underfitting

The term underfitting in supervised ML environment is used when a model is incapable of predicting training data as well as unseen data well enough. In this case, the model is incapable

of establishing the dominant trend within the data. This occurs when a model is too simple, which could be a result of the model needing more features, more degrees of freedom, less regularization or simply more training time [15]. Since an underfitting model cannot generalize to new data, the model won't be usable for classification tasks or prediction tasks.

Adapting an underfitting metric to unsupervised outlier detection models is a difficult task, since the ground truth of the data is not given without labels. However, there is another approach to measure underfitting in an anomaly detection scenario. Underfitting can be measured by comparing the detected outlier ratio in the training data and the test data. This procedure is shown in algorithm 1. The algorithm takes training and test data as inputs and computes for both data sets the ratio of outliers which is done by the function $compute\_outlier\_ratio(clf, data)$. Finally, the absolute difference of both ratios is computed as an indicator of underfitting. Notice that test data is a fraction of the training data which is separated before training the model. Therefore, test data is unseen data, but can be treated as training data for in the given scenario of an unsupervised outlier detection model.

Algorithm 1: Underfitting

Data: $train\_data, test\_data$
$ratio_{train} = compute\_outlier\_ratio(clf, train\_data)$
$ratio_{test} = compute\_outlier\_ratio(clf, test\_data)$
$score = \text{abs}(ratio_{train} - ratio_{test})$
return $compute\_score(score, threshold)$

In table 3.1 the underfitting trust score mappings are shown.

Underfitting Score Map	
$x = abs(ratio_{train} - ratio_{test})$	**trust score**
$0 \leq x < 0.01$	5
$0.01 \leq x < 0.025$	4
$0.025 \leq x < 0.05$	3
$0.05 \leq x < 0.1$	2
$0.1 \leq x$	1

Table 3.1: Underfitting Score Map.

3.4.2 Overfitting

Compared to underfitting models, overfitting models have low training error, but are also bad in predicting unseen data accurately. Overfitting is the opposite of underfitting, occuring when the model has been overtrained and contains too much complexity [15].

As we switch to the unsupervised approach of measuring overfitting, it is necassary to adapt certain aspects. Again, the ground truth of the training data is not given without labels. For this reason, we take another dataset at hand which contains only outliers. With this implicitly labeled outlier dataset, it is now possible to calculate the models overfitting score which is shown in algorithm 2.

The algorithm takes the training, test and outlier data sets as inputs together with the outlier percentage ($outlier\_percentage$) which is needed to compute how much the detected outlier ratio deviates from the real outlier ratio in normal behaviour data (training and test data). First the underfitting metric score is computed. Then the mean value of the outlier ratio in the outlier data set and the relative outlier detection accuracy in the test data is calculated which serves as an indicator of overfitting. Note that this last step is only computed when there is little to no underfitting (underfitting score ≥ 3). Otherwise the overfitting score is 1.

Algorithm 2: Overfitting

Data: $train\_data, test\_data, outlier\_data$
Input: $outlier\_percentage = 0.1$
$score_{underfitting} = compute\_underfitting\_score(clf, train\_data, test\_data)$;
if $score_{underfitting} \geq 3$ **then**
  $ratio_{outlier} = compute\_outlier\_ratio(clf, test\_data)$;
  $ratio_{test} = compute\_outlier\_ratio(clf, outlier\_data)$;
  $diff = \text{abs}(ratio_{outlier} - ratio_{test})$;
  $diff_{abs} = \text{abs}(outlier\_percentage - diff)/outlier\_percentage$;
  return $\text{mean}(ratio_{outlier}, diff_{abs})$;
else
  return 1;

Table 3.2 lists the score mappings for the overfitting metric. Note that overfitting is only computed when no to little underfitting occurs ($score_{underfitting} \geq 3$), otherwise the score of overfitting will be 1.

Overfitting Score Map	
$x = \text{mean}(ratio_{outlier}, diff_{abs})$	**trust score**
$1 \geq x > 0.95$	5
$0.95 \geq x > 0.9$	4
$0.9 \geq x > 0.8$	3
$0.8 \geq x > 0.75$	2
$0.75 \geq x$	1

Table 3.2: Overfitting Score Map.

3.4.3 Statistical Parity Difference

The population is represented as a set X with a known subset $S \subseteq X$ that represents the "protected" group. For example X might be a set of applicants that applied for an open position and S is the set of people that has tattoos. We are afraid that a firm's program that recommends inviting applicants could discriminate against people with tattoos by relatively inviting fewer people from the protected minority group compared to the unprotected majority of people with no tattoos. The discriminatory behaviour of the program's model can origin from the underlying training data with hidden bias. Computing statistical parity difference (SPD) provides information about whether the same ratio of samples belonging to the minority and majority receive a

favourable prediction ($\hat{Y} = 1$). $Pr(\hat{Y} = 1|P = p)$ represents the probability of receiving a favourable prediction, if the sample belongs to the protected minority $(P = 1)$ or the unprotected majority $(P = 0)$. SPD is then computed as shown in equation 3.1. The closer SPD comes to zero, the more a classifier is regarded as fair. Equation 3.2 presents perfect fairness as the ratio of positive outcomes for the majority and minority group is the same.

$$SPD(\hat{Y}, Y, P) = |Pr(\hat{Y} = 1 \mid P = 1) - Pr(\hat{Y} = 1 \mid P = 0)| \tag{3.1}$$

$$Pr(\hat{Y} = 1 \mid P = 1) = Pr(\hat{Y} = 1 \mid P = 0) \tag{3.2}$$

We now take a look at the approach of computing SPD for unsupervised outlier detection. Since one cannot take the labels from the training data set directly to compute SPD, we now must take the ratio of detected outliers from both groups and take the absolute difference. This fulfils the same purpose as the standard approach for supervised models, measuring whether more outlier are detected in a minority group compared to the majority group. This adjusted calculation represents a shift from the intended outcome to the actual outcome.

In the following algorithm 3 the calculation of SPD for unsupervised outlier detection is shown in more detail. The function takes training data as input and splits it into a protected and unprotected group regarding the defined protected feature and protected values. After that, the outlier detection ratios for both groups are computed. Giving that, the proportion of outliers for each group ($ratio_{(un)protected\_group}$) can be computed by dividing the number of detected outliers ($num\_outliers_{(un)protected\_group}$) by the number of total outliers ($num\_outliers_{total}$). In the last step, both proportions are taken to calculate SPD by measuring the absolute difference as in equation 3.1.

Algorithm 3: Statistical Parity Difference

Data: $train\_data$
$protected\_group = train\_data.get\_group(protected\_feature, protected\_values)$;
$unprotected\_group = train\_data \setminus protected\_group$;
$outliers = detect\_outliers(clf, train\_data)$;
$num\_outliers_{total} = outliers.count()$;
$num\_outliers_{protected\_group} = (outliers \cap protected\_group).count()$;
$num\_outliers_{unprotected\_group} = (outliers \cap unprotected\_group).count()$;
$ratio_{protected\_group} = num\_outliers_{protected\_group}/num\_outliers_{total}$;
$ratio_{unprotected\_group} = num\_outliers_{unprotected\_group}/num\_outliers_{total}$;
return $abs(ratio_{protected\_group} - ratio_{unprotected\_group})$;

Table 3.3 shows the trust score mappings for the SPD metric. The less statistical parity difference occurs $(abs(ratio_{protected\_group} - ratio_{unprotected\_group}))$ the higher score is given.

3.4.4 Disparate Impact

The disparate impact (DI) metric computes the ratio of samples from the protected group receiving a favourable prediction divided by the ratio of samples from the unprotected group receiving

Statistical Parity Difference Score Map	
$x = abs(ratio_{protected\_group} - ratio_{unprotected\_group})$	**trust score**
$0 \leq x < 0.01$	5
$0.01 \leq x < 0.025$	4
$0.025 \leq x < 0.05$	3
$0.05 \leq x < 0.075$	2
$0.075 \leq x$	1

Table 3.3: Statistical Parity Difference (SPD) Score Map.

a favourable prediction. This can be seen in equation 3.3. $Pr(\hat{Y} = 1|P = p)$ represents the probability of receiving a favourable prediction, if the sample belongs to the protected minority $(P = 1)$ or the unprotected majority $(P = 0)$.

$$DI(\hat{Y}, Y, P) = \frac{Pr(\hat{Y} = 1|P = 1)}{Pr(\hat{Y} = 1|P = 0)} \tag{3.3}$$

A value of 1 indicates parity for the protected feature. Values higher than 1 indicate a higher predicted positive outcome for the protected group. This is called a positive bias. A negative bias is present if the value is less than 1. An example of disparate treatment could be an employer using information about an applicant's vaccination status to exclude non-vaccinated applicants. Federal laws in the US prohibit such discrimination based on race, color, sex, sexual orientation, gender identity, national origin, religion, age, disability and genetic information. Because there may be correlation between features, it is not sufficient to simply exclude sensitive attributes from decision making. For example, removing sensitive demographic attributes from a training data set that still includes zip code as a feature may still have unfair treatment of subgroups, because zip codes can serve as a proxy for other demographic information.

Evaluating disparate impact towards unsupervised outlier detection models does not present major difficulties. The calculation depends on the predicted outcome, for which we can neglect the non-existence of labels. We now compute disparate impact as the ratio of outliers detected in the protected group and outliers detected in the unprotected group shown in algorithm 4. The algorithm differs in only two points to the statistical parity difference algorithm 3. Firstly, test data instead of training data is taken as input. And Secondly, in the last step the DI equation 3.3 is used instead of the SPD equation 3.1.

Algorithm 4: Disparate Impact

Data: $test\_data$
$protected\_group = test\_data.get\_group(protected\_feature, protected\_values)$;
$unprotected\_group = train\_data \setminus protected\_group$;
$outliers = detect\_outliers(clf, test\_data)$;
$num\_outliers_{total} = outliers.count()$;
$num\_outliers_{protected\_group} = (outliers \cap protected\_group).count()$;
$num\_outliers_{unprotected\_group} = (outliers \cap unprotected\_group).count()$;
$ratio_{protected\_group} = num\_outliers_{protected\_group}/num\_outliers_{total}$;
$ratio_{unprotected\_group} = num\_outliers_{unprotected\_group}/num\_outliers_{total}$;
return $abs(ratio_{protected\_group}/ratio_{unprotected\_group})$;

Trust score mappings for disparate impact are given in table 3.4.

Disparate Impact Score Map	
$x = abs(ratio_{protected\_group}/ratio_{unprotected\_group}))$	**trust score**
$1 \geq x > 0.95$	5
$0.95 \geq x > 0.9$	4
$0.9 \geq x > 0.8$	3
$0.8 \geq x > 0.75$	2
$0.75 \geq x$	1

Table 3.4: Disparate Impact (DI) Score Map.

3.4.5 Fairness Overview

An overview of the Fairness metrics is provided in table 3.5.

Fairness Metrics			
Metric	**Description**	**Dependencies**	**Condition**
Underfitting	Computes both the ratio of detected outliers in the training and test data and measures the absolute difference.	Model, training data, test data	-
Overfitting	Computes the mean value of the outlier ratio in the outlier data set and the relative outlier detection accuracy in the test data. Note that overfitting is only computed when there is little to no underfitting (underfitting score ≥ 3).	Model, training data, test data	An expected outlier percentage for the test data set must be defined.
Statistical Parity Difference	Computes the spread between the ratio of detected outliers in the protected and unprotected group given the training data set as input.	Model, training data, factsheet	A protected feature and protected values must be defined.
Disparate Impact	Divides the ratio of samples detected as outliers in the protected group by the ratio of samples detected as outliers in the unprotected group given the test data set as input.	Model, test data, factsheet	A protected feature and protected values must be defined.

Table 3.5: Overview Fairness Metrics

3.5 Explainability

Unsupervised machine learning provides a way to assess the massive amounts of unlabeled real-world data produced at a rapid speed. The majority of the current unsupervised ML techniques, however, do not offer a way for users to comprehend the underlying logic behind their decisions. These models serve as "black boxes" especially for those without domain expertise. Numerous negative effects result from this black box behavior, including limiting the user participation in improving the model and harming the user trust in these models, which prevents people from using them in real-world settings [22].

Receiving predictions from unsupervised ML models whether an anomaly exists or not is not enough for anomaly detection systems. It is important to have a justification for why it is an anomaly. We do need such information at hand to determine any potential catastrophic implications and make decisions to recover the system. Consequently, it is important to evaluate and explain the inner workings of these unsupervised ML models.

Different metrics are selected to deliver explanations about unsupervised outlier detection models and are described in the following part. The goal is to better understand decisions of AI based systems, what it has done, what it is doing currently and what will happen in the future.

3.5.1 Feature Correlation

This metric calculates how many features are highly correlated as described in algorithm 5. The input features should ideally not correlate with each other and strongly correlated features should have been removed. When attempting to represent feature contribution, biases are introduced by having a high correlation across features in most explanation methodologies. Partial dependence plots, for instance, are frequently used to demonstrate the significance of features by displaying the small marginal impact of a single feature on the model's anticipated outcome for supervised models. However, if two features are associated and one of them is permuted the other feature is still included in the model. The outcome would not significantly alter as a result, and the marginal effect would be biased. Due to the association, one of the feature would not be relevant. Additionally, many ML models assume that the variables are uncorrelated. If this assumption is incorrect, then previously interpretable models are no longer interpretable. In general, it is challenging to develop explanations for highly correlated features since it is impossible to separate the impacts of highly correlated features from each other, which results in biased explanations.

Algorithm 5: Feature Correlation

Data: $train\_data, test\_data$
Result: computes the percentage of highly correlated features
$data = \text{concat}(test\_data, train\_data)$
$correlation\_matrix = \text{correlation}(data)$
$upper\_matrix = \text{upper\_triangle}(correlation\_matrix)$
$num\_high\_occurences = 0$
for $corr$ ***in*** $upper\_matrix$ **do**
  if $corr \geq high\_corr\_thresh$ **then**
    $num\_high\_occurences = num\_high\_occurences + 1$
  end
  $perc\_corr\_feat = num\_high\_occurences / \text{num\_of\_features}(data)$
end
return $perc\_corr\_feat$

Table 3.6 contains the trust score mappings for the feature correlation metric. The more percentage of correlated features the worse the trust score results.

Feature Correlation Score Map	
$x = perc\_corr\_feat$	**trust score**
$x < 0.05$	5
$0.05 \leq x < 0.15$	4
$0.15 \leq x < 0.25$	3
$0.25 \leq x < 0.4$	2
$0.4 \leq x$	1

Table 3.6: Feature Correlation Score Map.

3.5.2 Model Size

The model size metric is based on the number of feature parameters a model takes as input and does not have to be adjusted for unsupervised models. This number of feature parameters correlates with the input dimension of models such as logistic regression or the number of edge weights and biases in neural networks and therefore have an impact on the complexity and comprehensibility of the underlying model. It is also assumed, that the comprehensibility inversely correlates with the models degrees of freedom, which is again dependent on the number of feature parameters. However, the model size metric does not capture any semantics, which means that as a stand-alone metric for explainability, it would not be enough [8]. As an example a big decision tree is still more interpretable than a small neural network. But when we take the same ML model and only vary the number of feature parameters, the model with smaller size is always more comprehensive. Therefore, the model size metric still contributes to measuring the level of explainability and can be taken into account.

Model Size Score Map	
$x = num\_of\_features$	**trust score**
$x < 10$	5
$10 \leq x < 25$	4
$25 \leq x < 50$	3
$50 \leq x < 100$	2
$100 \leq x$	1

Table 3.7: Model Size Score Map.

3.5.3 Permutation Feature Importance Score

The key idea of the permutation feature importance is to measure the importance of each feature individually by permuting its values. The increased prediction error by permuting the feature's values is taken as a measurement for the feature's importance. A feature is important if shuffling its values increases the prediction error, because the higher the observed error the more impact a feature must have on the prediction. Vice verca, if shuffled values of a feature leaves the prediction unchanged, the model does not rely on this feature for its prediction [16]. A big advantage of this algorithm is that you don't need to know the model's underlying classifier to calculate the importance of a feature. The model can thus be evaluated as a black box, with no details of the model and training data at hand.

The permutation feature importance algorithm was first developed for random forest classifiers and generalised for all classifiers by [7].

In algorithm 6 the computational details of the permutation feature importance metric is given. The function computes the importance of the features by iterating over each feature column, shuffling its column vector and measuring the error in the shuffled prediction outcome compared to to the non-shuffled prediction outcome ($err_{no\_shuffle} - err_{shuffled}$). The error indicated the importance of the feature. The metric also yields a features list sorted by the importance of each feature in descending order.

Algorithm 6: Permutation Feature Importance

Result: computes the importance of each feature
$dict_{scores} = \{\}$
$num\_redundant\_feat = 0$
$err_{no\_shuffle} = L(clf, data)$
for f ***in*** *features* **do**
  $data_{shuffled} = shuffle(f, data)$
  $err_{shuffled} = L(clf, data_{shuffled})$
  $imp_f = (err_{no\_shuffle} - err_{shuffled})$
  $dict_{scores}.append(imp_f)$
  if $imp_f == 0$ **then**
    $num\_redundant\_feat = num\_redundant\_feat + 1$
  end
end
return $\text{sort}(dict_{scores}, desc), num\_redundant\_feat/num\_feat$

Trust score mappings for the permutation feature importance metric are shown in table 3.8.

Permutation Feature Importance Score Map	
$x = num\_redundant\_feat/num\_feat$	**trust score**
$0 \leq x \leq 0.05$	5
$0.05 < x \leq 0.1$	4
$0.1 < x \leq 0.15$	3
$0.15 < x \leq 0.2$	2
$0.2 < x$	1

Table 3.8: Permutation Feature Importance Score Map

3.5.4 Explainability Overview

An overview of the Explainablility metrics is provided in table 3.9.

3.6 Robustness

Machine learning has proven to be a useful method for developing secure computer networks using anomaly detection-based intrusion detection systems. Data is essential for the creation of ML systems and can be targeted by hackers. In essence, one of the methods most frequently used to deceive ML models using data is data poisoning or contamination. These attackers aim to increase the classification errors in supervised ML environments or anomaly detection errors in unsupervised ML environments at test time [18].

Explainability Metrics			
Metric	**Description**	**Dependencies**	**Condition**
Feature Correlation	Computes the percentage of highly correlated features with a correlation reference value at hand.	Model, train data, test data	The reference value of correlation is set to 0.9 by default but can be set individually.
Model Size	Computes the size of the model by counting the number of features.	Test data	-
Permutation Feature Importance	Computes the importance of the features by iterating over each feature column, shuffling its column vector and measuring the error (= importance of the feature) in the shuffled prediction outcome compared to to the non-shuffled prediction outcome. The metric also yields a features list sorted by the importance of each feature in descending order.	Model, outliers data	-

Table 3.9: Overview Explainability Metrics.

A model is said to be robust if it can withstand exposure to unexpected data when performing the same task. The capabilities of anomaly detection models are threatened in real-world applications by data contamination and adversarial attacks. Therefore, it is necessary to evaluate ML models towards robustness before setting it up in real world.

3.6.1 CLEVER Score

The Cross Lipschitz Extreme Value for Network Robustness, in short CLEVER, is an extreme value theory approach [21]. Figure 3.1 illustrates the main intuition behind the Lipschitz constant estimation on robustness. The constant is used to estimate an upper bound on changes in the prediction outcome $g(x_0+\delta)$ with respect to input perturbations δ and x_0 being a known data point. An upper bound is given by $g(x_0)+L_q||\delta||_p$ and a lower bound is given by $g(x_0)-L_q||\delta||_p$, where L_q is the local Lipschitz constant and $||\delta||_p$ is the l_p norm of distortion δ with $p \geq 1$.

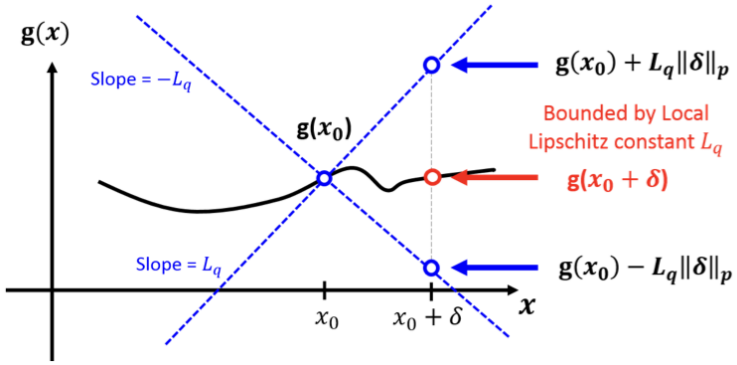

Figure 3.1: Intuition behind CLEVER [21].

In order to retrieve a neural networks local robustness, the minimal perturbation to change the prediction outcome is calculated by using the Lipschitz constant [21].

The CLEVER metric was introduced for classification models (supervised), but can be used for unsupervised anomaly detection models as well, since the implementation from the IBM art library [9] ($clever\_u$) does not depend on class labels.

3.6.2 Robustness Overview

In table 3.10 an overview of the Robustness metrics is provided.

Robustness Metrics			
Metric	**Description**	**Dependencies**	**Condition**
CLEVER	Computes the minimal perturbation that is required to change the prediction outcome.	Model	Can only be calculated on neural networks.

Table 3.10: Overview Robustness Metrics.

3.7 Methodology

The methodology pillar examines and validates the documentation and communication of significant decisions made during the life cycle of a ML model. The model training and validation process is described in the methodology score. It verifies whether the choices were consistent with accepted best practices. Having access to this knowledge is meant to increase users' confidence in the system.

The stability of predictions and stable outlier detection are increased by using the right data pre-processing techniques. Given this, we cannot exclude the impact of data pre-processing methods used for the calculation of the trust score. The end users' ability to understand the algorithm's specifics must also be taken into account. This can be measured by computing the completeness of the models' factsheet.

3.7.1 Normalization

Normalizing the data is one of the main pre-processing procedures for many statistical learning tasks. Because distinct features in a data set may utilize different measurement units, normalization is particularly crucial in unsupervised outlier detection [14].

Various normalizing techniques, including mean and standard deviation normalization, min-max normalization, and median normalization, are described in the literature. Normalization is a best practice as it is recognized to improve the performance of machine learning models [13]. It therefore is important to know whether a model was trained on normalized or non-normalized data.

Normalization Score Map	
Normalization technique	**Trust score**
Training and test data are standardized	5
Training data is standardized	4
Training and test data are normalized	3
Training data is normalized	2
None	1

Table 3.11: Overview Methodology Metrics.

3.7.2 Regularization

Due to the presence of input noise, the small size of the training set, and the complexity of classifiers, overfitting is a fundamental problem in supervised machine learning. One of the key methods of preventing machine learning algorithms from overfitting data is regularization [20]. Regularization techniques like Lasso, Ridge, and ElasticNet are just a few that can be applied. When deep neural networks are trained with millions of parameters, regularization is generally a need to avoid memorization. Regularization enhances the model's quality and is regarded as a best practice in the literature. In order to calculate a final trust score, it is necessary to take this metric into consideration that verifies the regularization method.

Regularization Score Map	
Regularization technique	**Trust score**
Elastic net regression	5
Lasso regression	4
Other	3
None	1
Not specified	Not computable

Table 3.12: Score Map Regularization.

3.7.3 Missing Data

The missing data metric computes the number of missing values in the training data and test data. A model trained on a dataset with many missing values cannot be reliable. Also evaluating the model with test data containing missing values can lead to inaccuracies in the test metrics.

A common approach is to fill missing values with mean values, which is achieved by the missing-indicator method. Even though this method is commonly used, it still creates a bias on prediction [5].

Another method is to fill missing values with random values, which would disturb the statistical properties of the data and is therefore regarded as a bad practice. It is difficult to trust a model, if missing values are existent or filled with bad practice methods. The missing data metric thus comes well at hand.

Missing Data Score Map	
Missing data	**Trust score**
No missing values	5
Missing values	1

Table 3.13: Missing Data Score Map.

3.7.4 Train Test Split

Creating training and testing splits from the data is important. If the training split does not contain enough data, the performance of the model will not be sufficient. If the test split does not contain enough data, it cannot represent the overall data, and the values of the performance metric for the test data cannot be generalized. For example, the test data may be insufficient to accurately represent the performance of the data if the model is trained on 95% of the data and tested on only 5% of it. Therefore, the split proportion must be chosen carefully. The split size not only affects the performance, it also determines whether the performance metrics are still applicable. Therefore, this is an important metric to consider when calculating trust scores.

In listing 3.1 the mappings for a trust score are shown. A train test split of 80% to 20% is well known and is often used as the state-of-the-art split and therefore receives a trust score of 5. The further off the train test split is from this reference point the worse the trust score becomes.

Listing 3.1: Train Test Split Score Map

```
{
    "50-60 95-97": 1,
    "60-75 90-95": 2,
    "70-75 85-90": 3,
    "75-79 81-85": 4,
          "79-81": 5
},
```

3.7.5 Factsheet Completeness

Factsheets are a necessity for the end user to assess trustworthiness and specifically methodology of an AI-based system. It summarizes key meta-information about a trained machine learning model and includes information about the structure, the creator, the goal and the data used of a model. The completeness of the factsheet measures whether the factsheet contains all the necessary information that users need (see algorithm 7). It is impossible to trust a model if we are not informed about the structure of the model.

Algorithm 7: Factsheet Completeness

Input: $general\_inputs, factsheet$
Result: computes the completeness of the factsheet
$n = \text{len}(general\_inputs)$
$counter = 0$
for e ***in*** $general\_inputs$ **do**
  if "$general$" ***in*** $factsheet$ ***and*** e ***in*** $factsheet$ **then**
    $counter = counter + 1$
  end
end
return $\text{round}(counter/n * 5)$

3.7.6 Methodology Overview

An overview of the Methodology metrics is provided in table 3.14.

Methodology Metrics			
Metric	**Description**	**Dependencies**	**Condition**
Normalization	Takes the normalization technique used to calculate the metrics score.	Training data, test data, factsheet	-
Regularization	Takes the regularization technique to calculate the metris score.	Training data, test data, factsheet	-
Missing Data	Counts the number of missing values in the training and test data.	Training data, test data, factsheet	-
Train Test Split	Takes the ratio of training data to test data to calculate the metrics score.	Training data, test data, factsheet	-
Factsheet Completeness	Computes the completeness of required properties in the factsheet by counting the ratio the required properties.	Training data, test data, factsheet	-

Table 3.14: Normalization Score Map.

Chapter 4

Webapp

4.1 Trusted-AI Webapp

For the last task in the scope of this bachelor thesis an existing webapp was extended. The web application called Trusted-AI was implemented by three former master students as part of their master's thesis. The webapp was created to evaluate models not only by their performance, but also by their trustworthiness. The trust score can supplement a solely performance-based evaluation when comparing different models. When choosing pretrained models, trustworthiness may be a key consideration depending on the application scenario. Monitoring the evolution of a model's trustworthiness over time might also help to spot potential improvements.

So far, the webapp has the ability to evaluate only supervised models. The main contribution of this bachelor thesis was to extend the given webapp and design and implement the underlying algorithm quantifying the trustworthiness for unsupervised models. This new algorithm is named Trusted-Anomaly-Detection. In order to design an algorithm that can quantify the trustworthiness of unsupervised models a taxonomy containing the most important metrics for trust was created. The theoretical background of these metrics can be found in chapter 3. The webapp allows the user to interact with the algorithm and analyse the outcomes of the metrics with provided details. The details are provided as different types of graphs and textually written explanations of all the different metrics.

4.2 Algorithm Design

To understand the design and implementation for unsupervised solutions, it makes sense to first take a closer look on the existing webapp and the underlying algorithm for supervised solutions:

The Trusted-AI algorithm evaluates over twenty different metrics while taking into account the four trust pillars of fairness, explainability, robustness and training methodology. The algorithm's main concept is that it accepts a machine learning model, its training and testing data, and the most crucial meta-data encoded in a factsheet as inputs. The algorithm then calculates

all applicable metrics for each pillar and combines the results to get a final trust score [12]. Below is a detailed explanation of the technique and design of the algorithm for supervised solutions and its application for unsupervised solutions.

Pillars

Four different pillars of trust have been selected from the literature which include fairness, explainability, robustness and methodology for supervised solutions [12].

During literature research for trustworthiness in unsupervised machine learning the same pillars of trust have crystallized out. Therefore, the pillar of fairness, explainability, robustness and methodology have stayed the same and been selected for the algorithm design and implementation for unsupervised solutions.

Taxonomy

Trusted-AI proposes several different metrics for each pillar and are shown in the following graph:

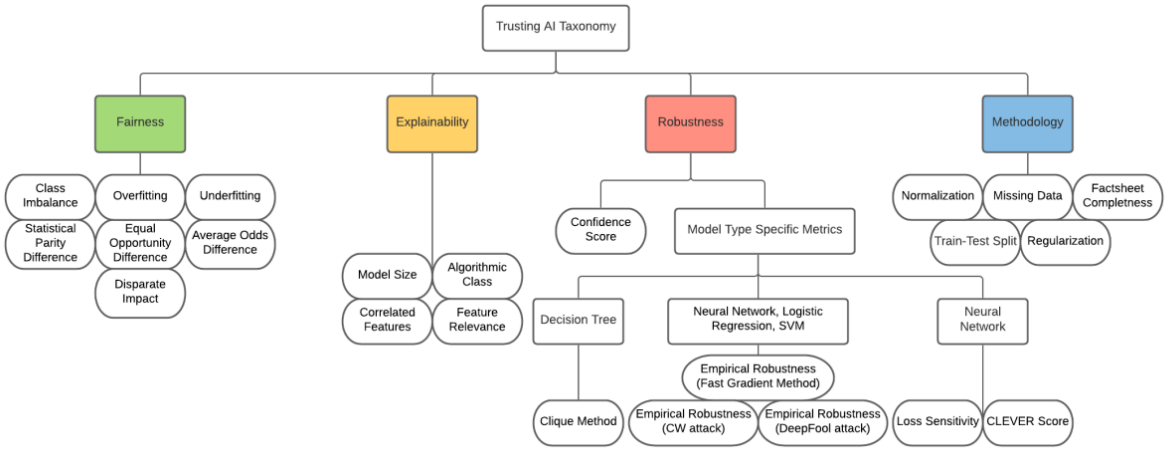

Figure 4.1: Trusted-AI [12]: Taxonomy.

All of these metrics have been analysed and tested as candidates to evaluate unsupervised solutions towards trustworthiness. Many of the metrics were not described for unsupervised solutions in the literature and had to undergo major adjustments (see chapter 3 for more details) to make it suitable for unsupervised solutions. Other metrics relied on labels and simply were not possible to adopt. All applicable metrics have been implemented in the final algorithm for unsupervised solutions and are listed in the tabular below. Note that the permutation feature importance metric does not originate from the Trusted-AI algorithm for supervised solutions.

Metrics for Unsupervised Solutions			
Fairness	**Explainability**	**Robustness**	**Methodology**
Underfitting	Correlated Features	CLEVER	Normalization
Overfitting	Model Size		Missing Data
Statistical Parity Difference	Permutation Feature Importance		Regularization
Disparate Impact			Train Test Split
			Factsheet Completeness

Table 4.1: Metrics for unsupervised solutions.

Parameters

Trusted-AI takes several artefacts as input parameters in order to calculate each pillars metrics and out of that the final trust score.

1. **ML model:** machine learning model
2. **Training data:** training data on which the given model was trained on
3. **Test Data:** test data to evaluate the model with non-seen data
4. **Factsheet:** containing the metadata of the model

All of these parameters are taken at the top level of the Trusted-AI algorithm and handed over to the next lower level, the pillars (fairness, explainability, robustness, methodology). Finally, the artefacts are handed over as input parameters to each metric for its computation. Each metric is implemented as a separate function and returns the metric value which is mapped to a trust score between 1-5. A trust score of 5 represents the highest trust related to a metric and the opposite is valid for a trust score of 1. On the final step, the Trusted-AI algorithm aggregates all individual metric trust scores into a final combined trust score.

The algorithm for unsupervised solutions works exactly the same but with an additional artefact as input parameters. The outlier data set. Note that this implementation focuses on unsupervised anomaly detection scenarios and is not applicable to all unsupervised machine learning scenarios.

1. **ML model:** machine learning model
2. **Training data:** training data on which the given model was trained on
3. **Test Data:** test data to evaluate the model with non-seen normal data
4. **Outlier Data:** outlier data to evaluate the model with non-seen outlier data
5. **Factsheet:** containing the metadata of the model

Weights and Configurations

As mentioned before, Trusted-AI takes each metrics trust score to compute the final trust score. Figure 4.2 illustrates the aggregation process, which demonstrates how the algorithm creates a final trust score from the separate metric trust scores.

Each of the four pillar is paired with a set of metrics, and within each set, each metric has a weight assigned in order to create a weighted average of the metric scores. Prioritizing more relevant metrics is made possible by the weights. It is debatable if each metric is equally significant and how much each one should differ in weight. Every metric has default weights that represent their value in the broadest sense (listing 4.1).

Listing 4.1: Default weights of metrics.

```
{
    "fairness": {
        "underfitting": 0.35,
        "overfitting": 0.15,
        "statistical_parity_difference": 0.15,
        "disparate_impact": 0.1
    },
    "explainability": {
        "correlated_features": 0.15,
        "model_size": 0.15,
        "permutation_feature_importance": 0.15
    },
    "robustness": {
        "clever_score": 0.2
    },
    "methodology": {
        "normalization": 0.2,
        "missing_data": 0.2,
        "regularization": 0.2,
        "train_test_split": 0.2,
        "factsheet_completeness": 0.2
    },
}
```

For each pillar individually, the weighted average of the metric scores is determined, and this yields an overall trust score for each pillar. Similar to calculating pillar scores, the final trust score is computed. The four pillars are given weights (listing 4.2), and the final trust score is the weighted average of those scores [12].

Listing 4.2: Default weights of pillars.

```
{
    "pillars": {
        "fairness": 0.25,
        "explainability": 0.25,
```

```
        "robustness": 0.25,
        "methodology": 0.25
    }
}
```

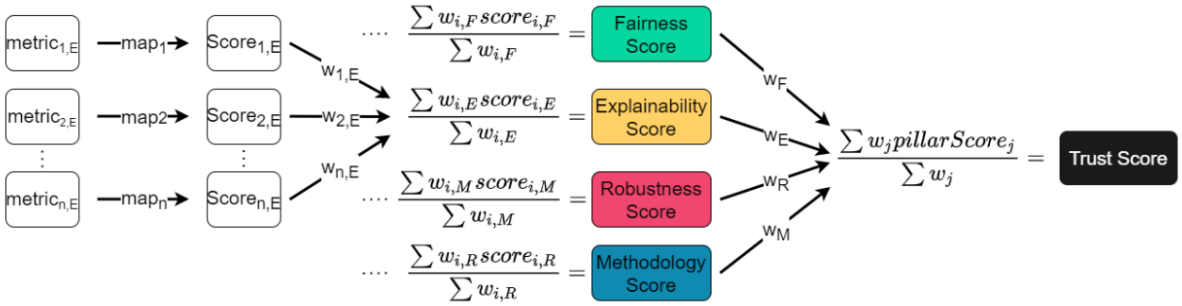

Figure 4.2: Trusted-AI: Final trust score computation [12]

Since the same approach can be used computing the final trust score for unsupervised solution does not present major difficulties. The implementation follows the same implementation as for supervised solutions.

Algorithm 8: Trusted Unsupervised Anomaly Detection Algorithm

Result: computes the final trust score for unsupervised anomaly detection solutions
Input: $model, trainingdata, testdata, outlierdata, factsheet, config_{map}, config_{weights}$
$trust\_score = 0$
$scores\_pillars = \{\}$
for *p* **in** *pillars* **do**
  $score_p = 0$
  $scores\_metrics = \{\}$
  $metrics = get\_metrics(p)$
  for *m* **in** *metrics* **do**
    $args = config_{map}[m]$
    $scores\_metrics[m] = get\_score_m(args)$
  end
  for $(m \in keys, v \in values)$**in** $scores\_metrics$ **do**
    $w_m = config_{weights}[m]$
    $score_p = score_p + w_m * v$
  end
  $scores\_pillars[p] = score_p$
end
for $(p \in keys, v \in values)$**in** $scores\_pillars$ **do**
  $w_p = config_{weights}[p]$
  $trust\_score = trust\_score + w_p * v$
end
return $trust\_score$

In algorithm 8 the computation of the final trust score for unsupervised solutions is shown in more details. The $model, trainingdata, testdata, outlierdata, factsheet, config_{map}$ and $config_{weights}$ are received as input parameters. The algorithm iterates over all pillars and computes each associated metric separately. For a given metric all the arguments are extracted from the $config_{map}$ and after that the computed score is stored in the $scores\_metrics$ dictionary. As soon as every metrics trust scores for for a given pillar is computed and stored the algorithm iterates over the $scores\_metrics$ dictionary and adds the weighted scores according to the $config_{weights}$. This added value corresponds to the aggregated trust score for a given pillar and is stored in the $scores\_pillars$ variable. In the last iteration the final $trust\_score$ is computed by adding the scores of each pillar which are weighted according to the $config_{weights}$.

4.3 Webapp Design

There are two different kinds of modules: upload modules, which let the user create scenarios and upload ML solutions for those scenarios to the server; and analyze modules, which let users get trust analyses for specific solutions that are calculated using the Trusted-AI algorithm and compare various solutions to each other. The analysis is shown graphically and may be interactively customized and thoroughly investigated. The suggested Trusted AI Algorithm is placed in the backend along with a database that stores the ML models and configurations.

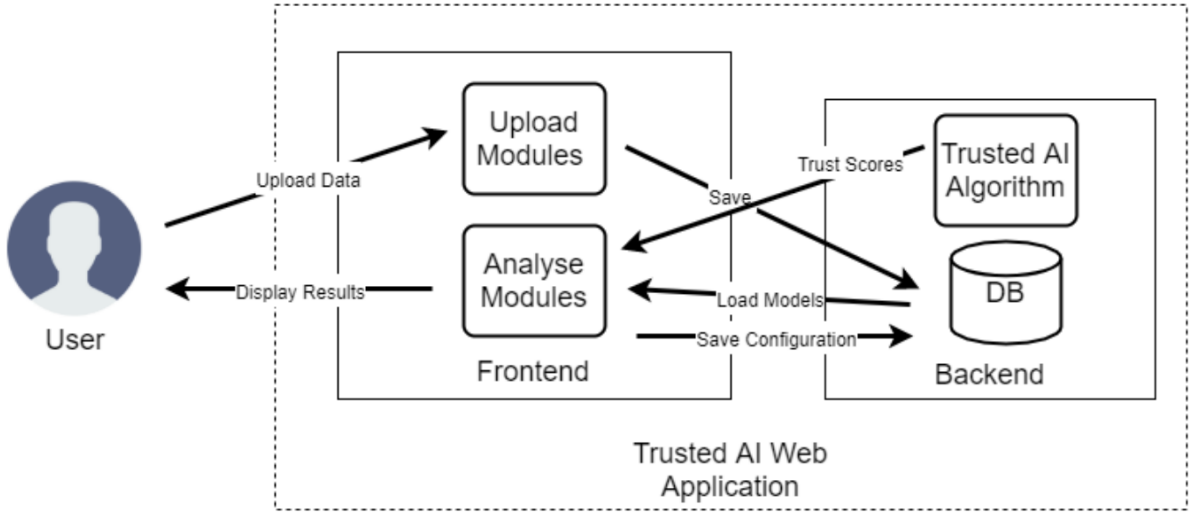

Figure 4.3: Webapp: Architecture
[12]

The Scenarios and Upload page (upload modules) and the Analyze and Compare page (analyze modules) make up the application's original navigation bar.

Figure 4.4: Webapp: Original Navigation Bar.

A toggle to switch the mode from supervised to unsupervised has been newly implemented in the navigation bar. The selected mode is saved globally in the application and is therefore applied to all pages. The default mode of the application is the supervised mode (4.5) and without switching the mode the application can be used exactly as before.

Figure 4.5: Webapp: Navigation Bar (in supervised mode).

As soon as the toggle is clicked the application switches to unsupervised mode where the differences and extensions to the old application become visible (see figure 4.6).

Figure 4.6: Webapp: Navigation Bar (in unsupervised mode).

The user has the option to specify and create a scenario on the initial scenarios page, as well as view previously saved scenarios. The user has the option to upload solutions for the specified scenario and to add further details about the solution on the following upload page. The user can then evaluate the trust analysis for a specific solution on the analyze page and modify the configuration of the algorithm. The whole trust analysis for the solution is also available to users as a PDF download. On the final compare page, the user can compare the trust ratings of other solutions [12].

4.4 Extension of Webapp for Unsupervised Anomaly Detection

4.4.1 Scenarios Page

On the scenarios page a user has the overview of all currently applicable scenarios and its model solution. One supervised scenario for example is the credit card approval which is a classification problem that separates credit applicants into approved or disapproved applicants. For this given scenario different classification models (solutions), such as random forest classifier, support vector machine etc. are listed beneath (see figure 4.7. The user also has the ability to upload new solutions for a given scenario which can then be analyzed on the other pages.

Figure 4.7: Scenarios page: Supervised scenarios.

Within the scope of this bachelor thesis, the scenarios page was extended to list also unsupervised scenarios with different solutions 4.8. When no unsupervised scenarios are uploaded by the user yet, the “IT security outlier detection scenario” is listed by default with different solutions to it.

Figure 4.8: Scenarios page: Unsupervised scenarios

Additional, it is now possible to add a new unsupervised scenario by clicking on the plus icon next to the title “SCENARIOS UNSUPERVISE”.

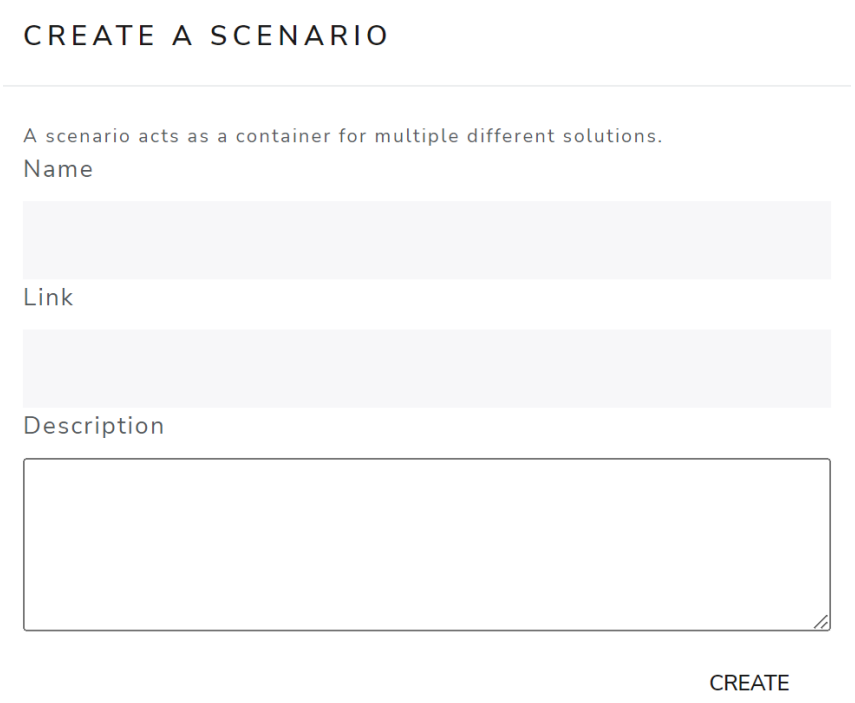

Figure 4.9: Scenarios page: Create new unsupervised scenario.

After the form is filled out with name, link to the data set and description the create button can be clicked and the new scenario is listed on the scenarios page as shown in figure 4.10.

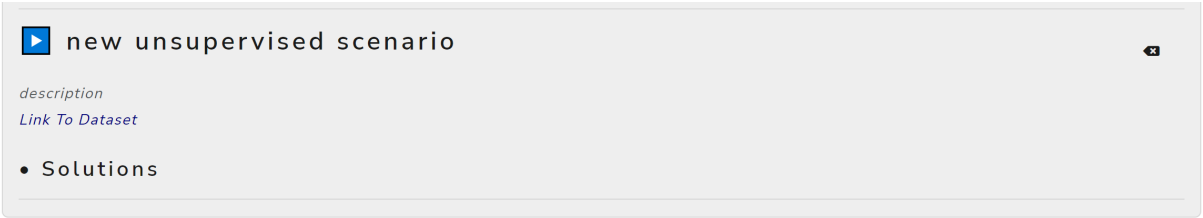

Figure 4.10: Scenarios page: New unsupervised scenario.

It is also possible to delete the newly added scenario by clicking on the delete icon seen on figure 4.10.

4.4.2 Upload Page

On the upload page for supervised solutions a user can pick a given scenario and upload a new solution to it. A user then has to provide the solutions name and description, upload the training data and test data, as well as specifying further details. These details contain protected features, protected values target column and favourable outcomes. As a last step the user has to upload a factsheet belonging to the specified solution (.json file) or create a new factsheet directly on the upload page. The factsheet again contains several details and is together with the other uploaded material necessary to do the final trust score calculation. The factsheet holds information about the developer, purpose and name of the model and contact information.

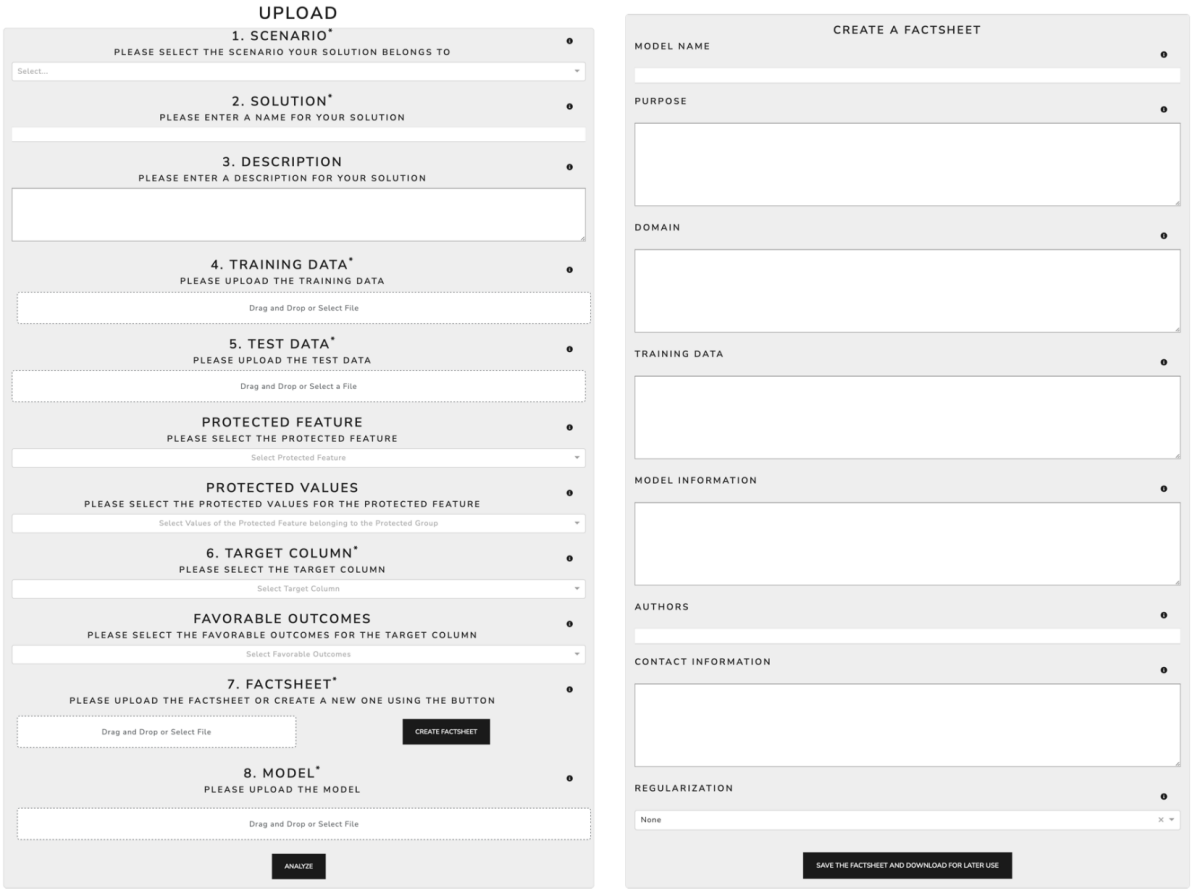

Figure 4.11: Upload page: Upload supervised solution.

The information given on the upload page is used to display the automated trust report on the analyze page. Other information such as protected features or protected values is used to compute certain metrics.

When switching into the unsupervised mode by clicking on the toggle in the navigation bar 4.6 the upload page updates its form such that a user can upload a new solution for a chosen anomaly detection scenario 4.12. Note that the scenario input field (1. SCENARIO) does only list anomaly detection scenarios to select. The mandatory outlier upload field (6. OUTLIER DATA) appears as this data set is necessary to compute several metrics. Also, the target column and favorable outcomes field disappear as unsupervised solutions do not have target columns in their respective data sets and therefore also no favourable outcomes to select. The implementation of creating a factsheet has not been changed as this does not require modification.

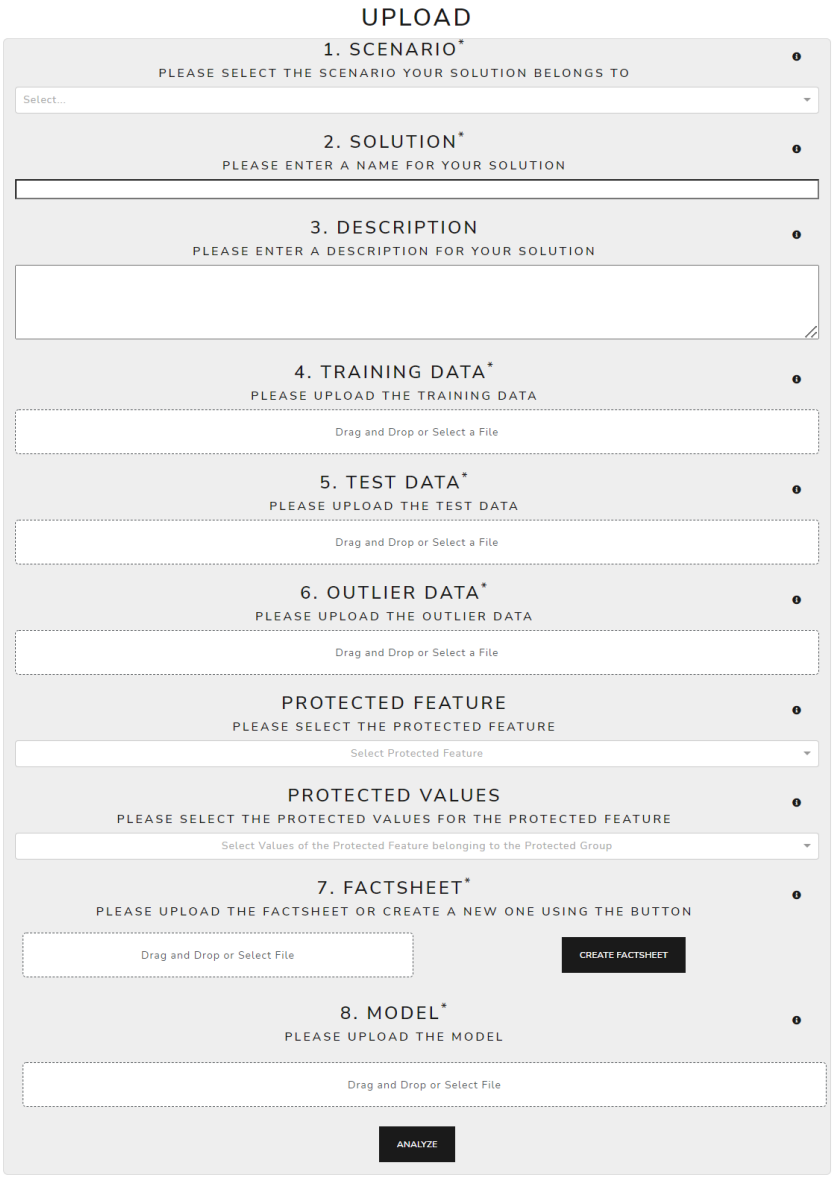

Figure 4.12: Upload page: Upload unsupervised solution.

4.4.3 Analyze Page

On the analyze page the user can select a given scenario and choose the solution to analyze. As soon as both are selected, the application triggers the Trusted-AI algorithm and calculates all pillars with its metrics for the selected solution. A report is visible and consists of two parts: Firstly, the general information that illustrates the chosen solution with scenario description and model information and secondly, the trustworthiness report which shows the trust scores of the algorithm. By navigating to the different pillars further details of the metrics scores appear.

There have been major changes to the analyze page to make it work for unsupervised solutions. If the unsupervised mode is active, both the scenarios and solutions drop-down only list

unsupervised scenarios and solutions. By selecting a solution as seen on figure 4.13 the webapp triggers the newly implemented Trusted-Anomaly-Detection algorithm for the trust scores calculations.

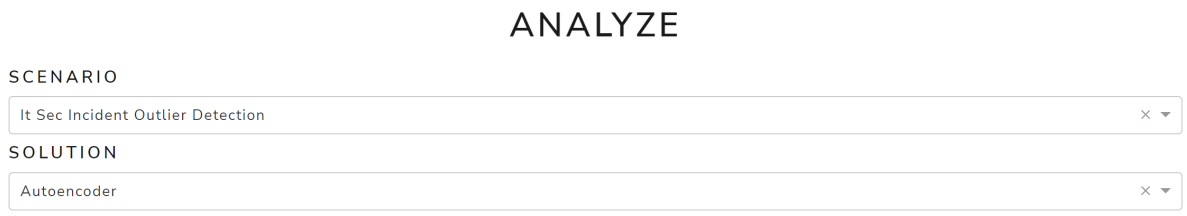

Figure 4.13: Analyze page: Select unsupervised scenario and solution.

As soon as the scores are available the report is shown on the page below consisting of the general information and trustworthiness report:

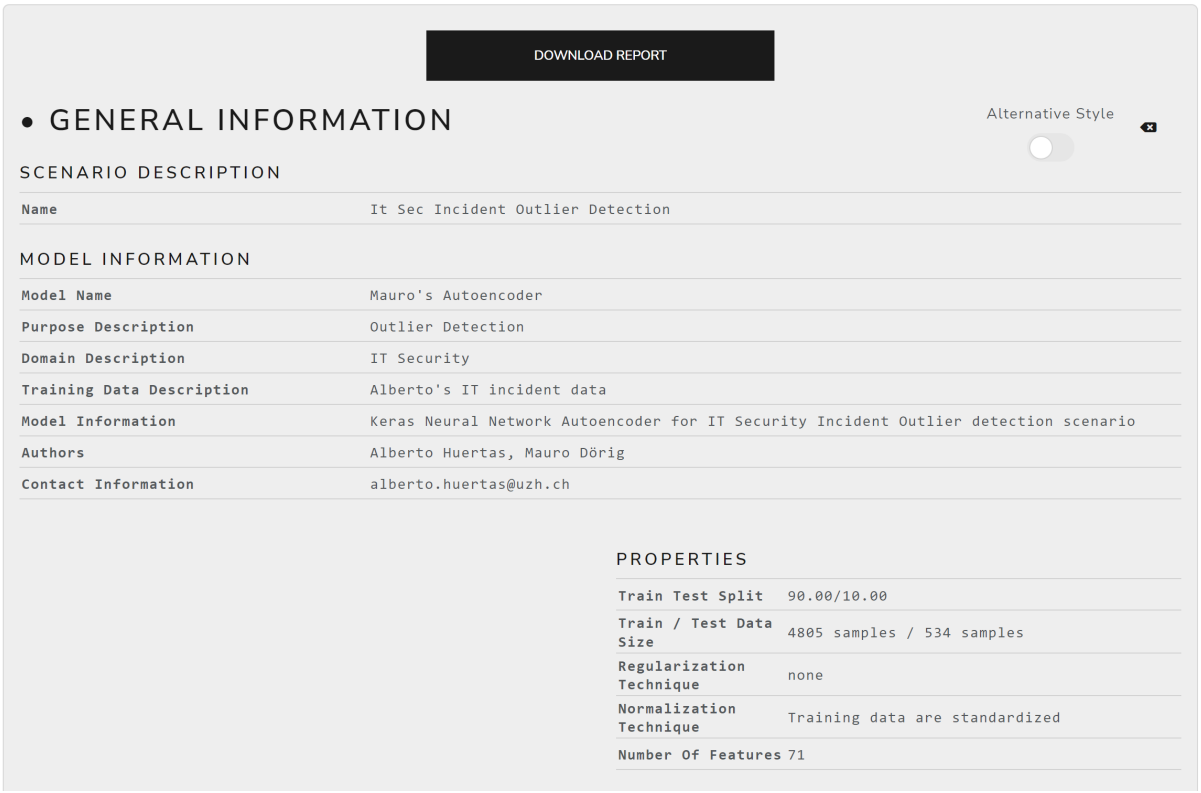

Figure 4.14: Analyze page: General information.

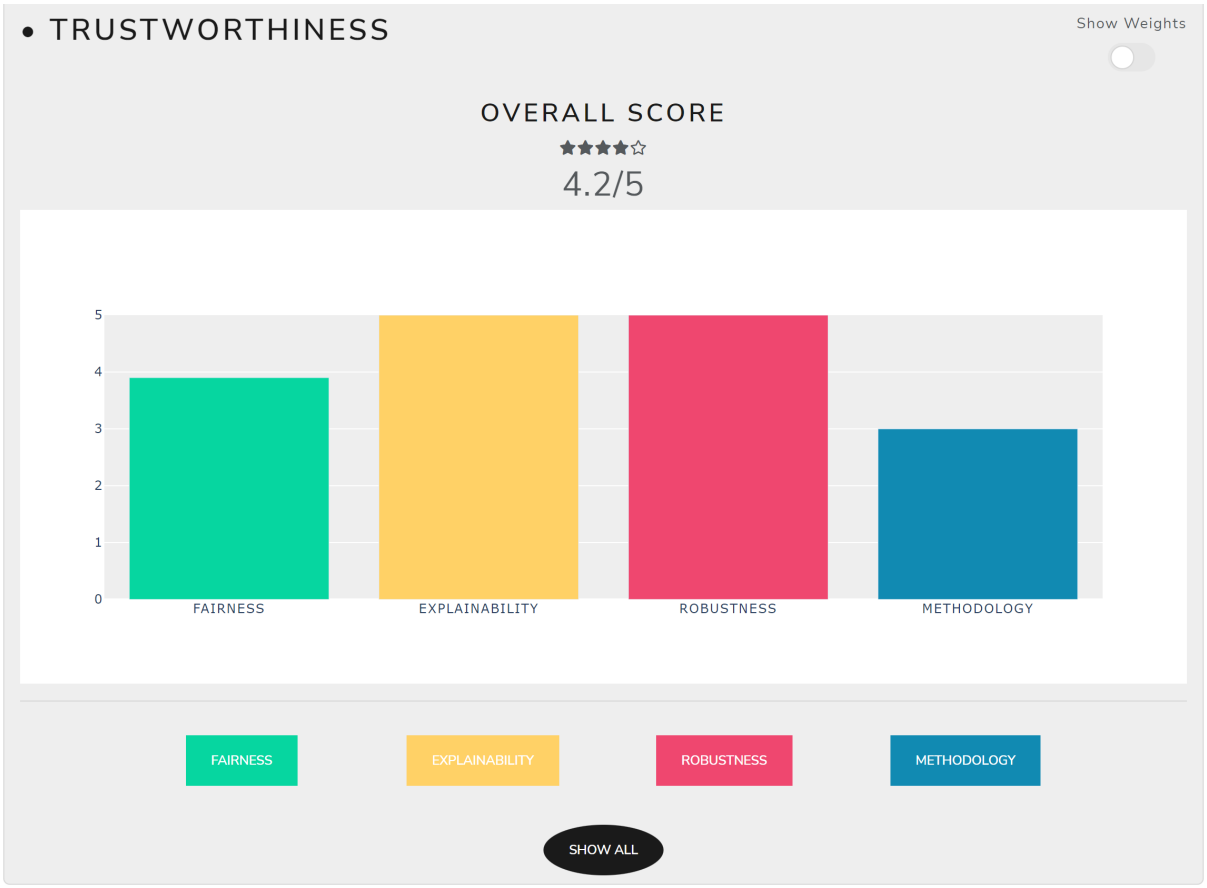

Figure 4.15: Analyze page: Trustworthiness report.

Major changes are applied on the given Figures above (4.14, 4.15). Firstly, the before used performance metrics are not possible to calculate for unsupervised solutions, since there are no labels in the data sets. For that reason, performance metrics are not shown. Secondly, scenario description and model information in the general information part had to be adopted for unsupervised solutions. Lastly, in the trustworthiness report, each pillars metrics for supervised solutions were replaced by metrics for unsupervised solutions. This includes also changes in the given details for each metric.

If for example the methodology pillar is selected by clicking on the green fairness button shown in figure 4.15 more details on the used metrics appear containing a graph (figure 4.16) which shows the scores for each metric and details on every computed metric (figure 4.17).

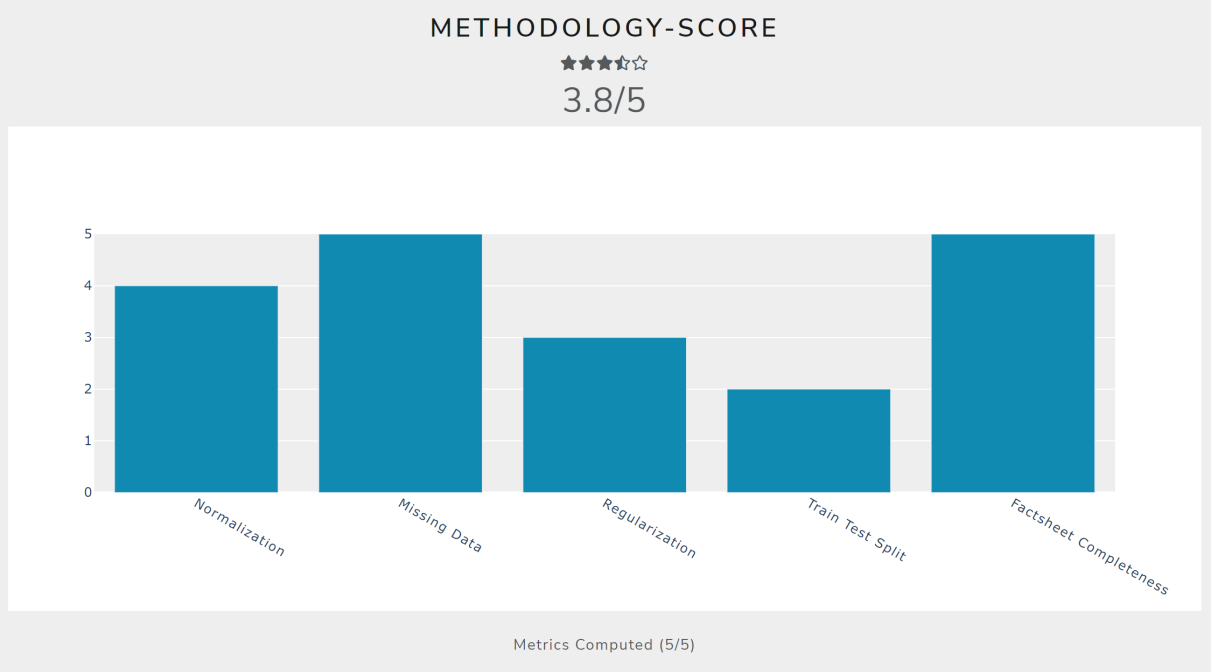

Figure 4.16: Analyze page: Methodology metrics scores.

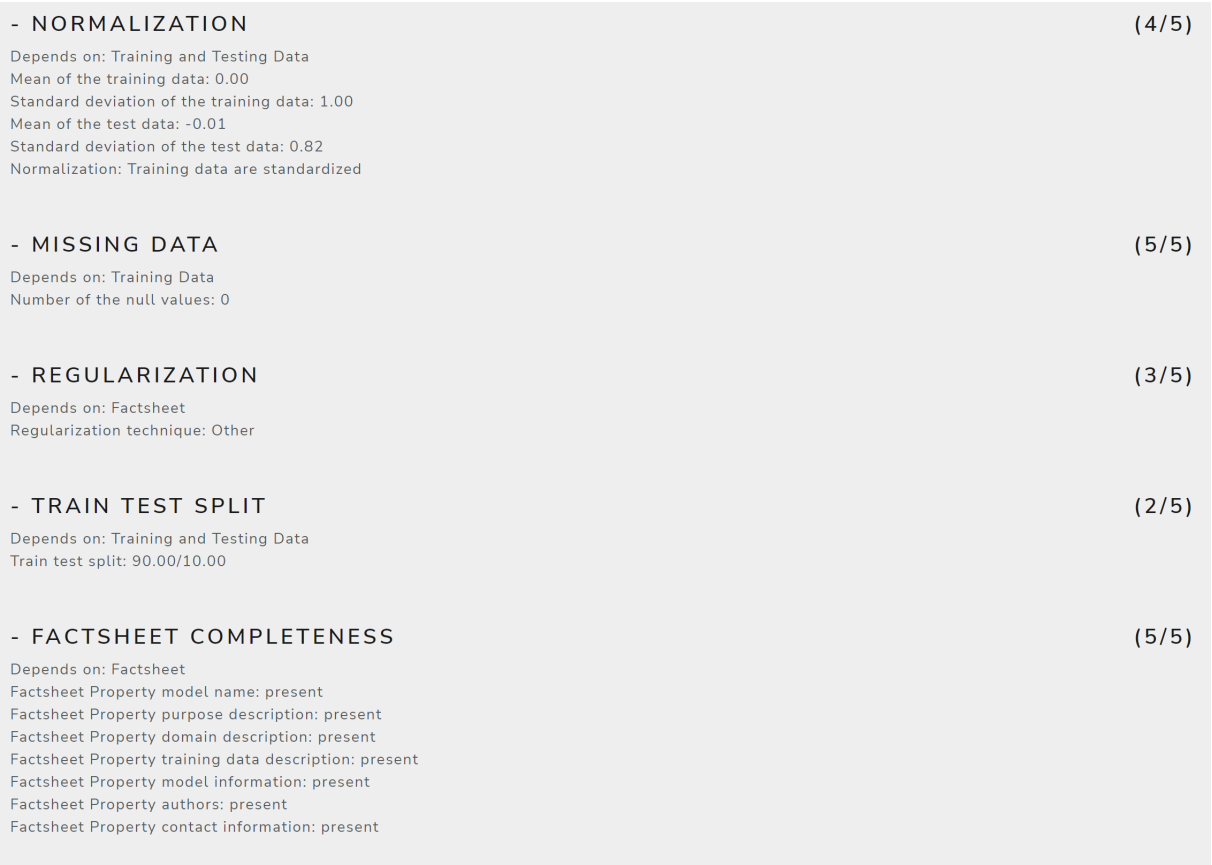

Figure 4.17: Analyze page: Methodology metrics details.

4.4.4 Compare Page

On the compare page a user can select two different solutions within a selected scenario and do a side-by-side comparison. The page shows both solutions trust reports which allows the user to compare each metrics outcome to a high level of details.

Changes on this side were similar to the changes on the analyze page. As soon as the unsupervised mode is activated a user can select a anomaly detection scenario from the drop down menu and choose two different solutions.

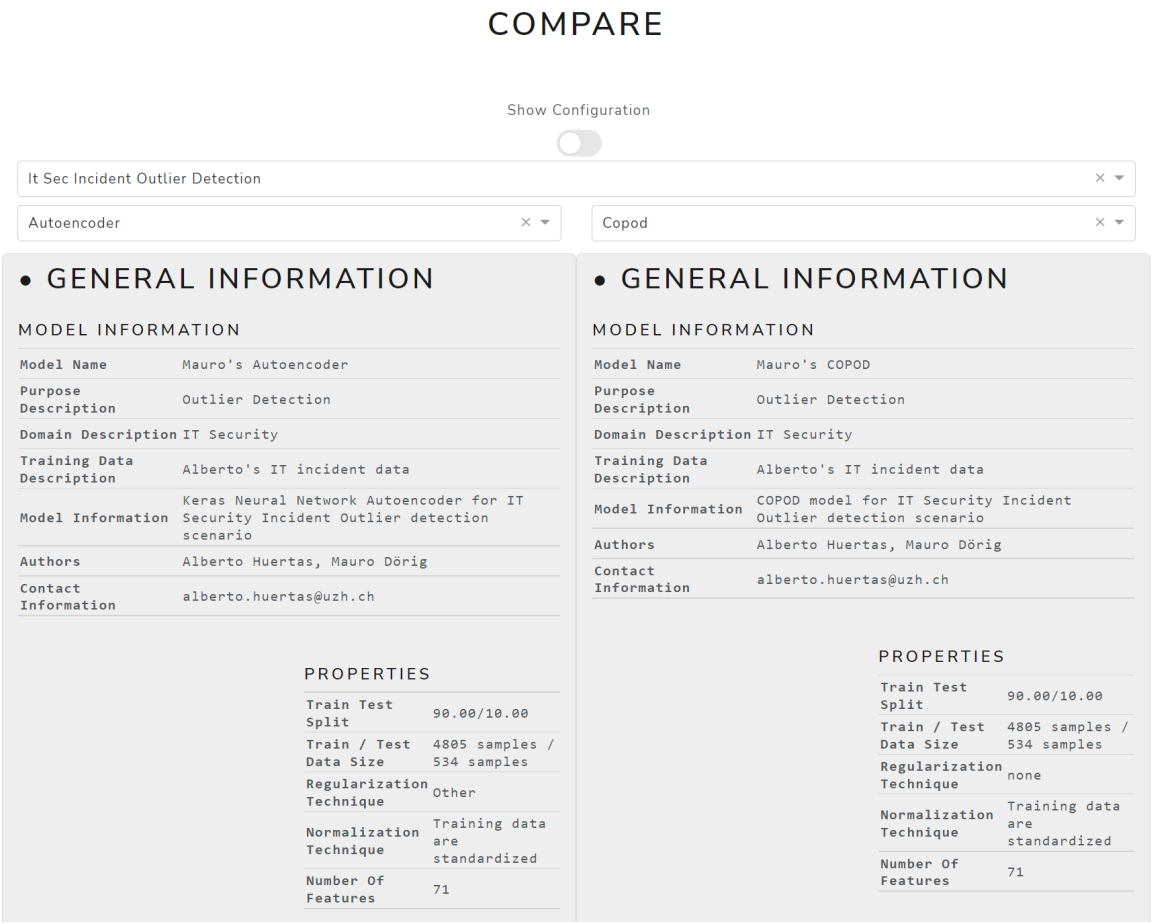

Figure 4.18: Compare page: Select and compare two solutions for a scenario.

By scrolling further down the page figures for each trust pillar appear and let the user compare each pillar and metric scores for both solutions side by side (figure 4.19). Note that the outcomes on figure 4.19 do not correspond to the selected solutions on figure 4.18 and are plotted for demonstration purposes.

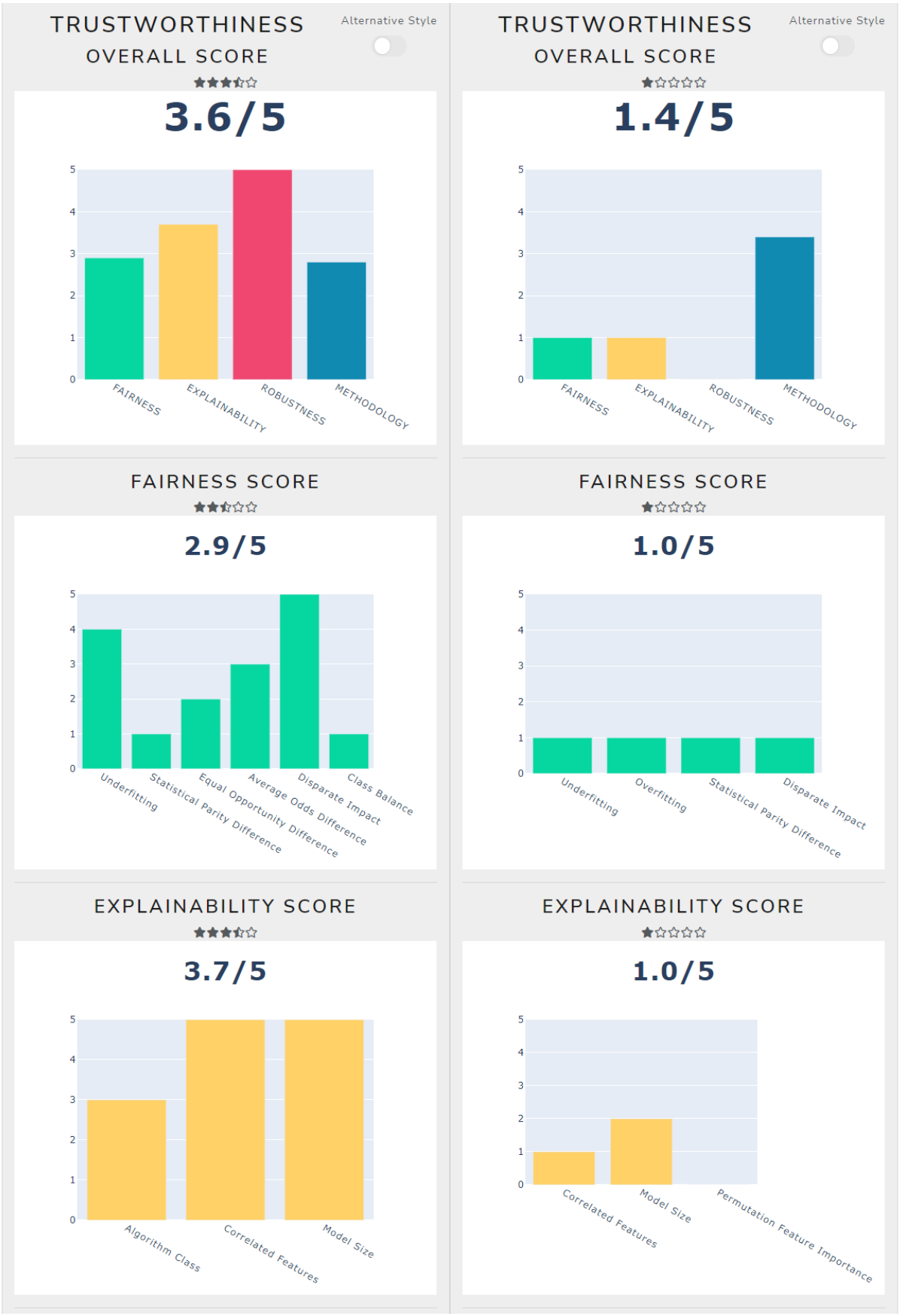

Figure 4.19: Compare page: Comparison of two selected solutions.

Chapter 5

Evaluation

In order to evaluate the trusted AI algorithm for unsupervised solutions a scenario detecting outliers in IT data was chosen which serves as an ideal example as this scenario is deeply concerned with the pillars of trust, namely fairness, explainability, robustness and methodology. Multiple solutions to this scenario were created and are described in the following parts.

5.1 Scenario: IT Security - IoT Data Anomaly Detection

Anomaly detection in the field of IT security is a common practice in real-life as it protects enterprises against cyber attacks by monitoring anomalous user behaviour on different kinds of IT systems. An anomaly detection solution can identify a user behaving abnormally to then take appropriate actions to limit users' access to the system.

This scenario is focused on detecting anomalies in data collected by a Raspberry Pi 3. The data set simulates how the Raspberry Pi behaves on the inside with regard to the network, CPU, RAM, tasks, the number of instructions executed, cache failures, etc. Data from the Linux Perf tool has been collected from when the Raspberry Pi is operating normally and when it has been infected with 6 separate attacks. Around 4000 data samples with 72 features are present. Different models are trained to detect anomalies in the monitoring data referred to as solutions.

5.2 Solutions: IT Security - IoT Data Anomaly Detection

Beneath every trained model lies the model's algorithm which is trained for different scenario cases to decide whether a data point is an anomaly or not. Different algorithms were selected for this scenario's case and are named as following:

1. Isolation Forest (IF)
2. Local Outlier Factor (LOF)

3. Learning One-Class Support Vector Machine (OSCVM)
4. Autoencoder
5. Cluster-Based Local Outlier Factor (CBLOF)
6. Lightweight On-Line Detector of Anomalies (LODA)
7. Copula-Based Outlier Detection (COPOD)
8. Lightweight On-Line Detector of Anomalies (LODA)
9. Histogram-Based Outlier Score (HBOS)
10. Single-Objective Generative Adversarial Active Learning (SO_GAAL)
11. Multi-Objective Generative Adversarial Active Learning (MO_GAAL)

5.3 Model Analysis

In the following section, different solutions for the given anomaly detection scenario are selected from above and analyzed. The following three models, Isolation Forest, Local Outlier Factor and Learning One-Class Support Vector Machine were trained by Alberto Huertas and handed to me to evaluate the webapp with the underlying Trusted-Anomaly-Detection algorithm. The same training, testing and outlier data sets were used in order to have an insightful comparison between the models. Also, the same train-test split $89\%|11\%$ were applied and no regularization nor normalization technique were used for all the models. The properties are shown in figure 5.1 and are valid for all the solutions.

PROPERTIES

Train Test Split	89.00/11.00
Train / Test Data Size	4496 samples / 550 samples
Regularization Technique	None
Normalization Technique	None
Number Of Features	67

Figure 5.1: Analyze page: Properties.

Before evaluating the given models towards their trustworthiness the outlier detection ratios on different data sets are discussed 5.1. The outlier detection ratio on outlier data sets is the only applicable performance metric for unsupervised anomaly detection models. Other performance

Outlier Detection Ratio			
Data set	**IF**	**LOF**	**OCSVM**
Normal behaviour: training data	5.0% (225/4496)	5.0% (225/4496)	4.72% (212/4496)
Normal behaviour: test data	12% (66/550)	13.82% (76/550)	11.82% (65/550)
Backdoor behavior: ssh+commands	100% (116/116)	100% (116/116)	100% (116/116)
Backdoor behavior: dataleak 1s	26.67% (16/60)	100% (60/60)	100% (60/60)
Backdoor behavior: dataleak 50s	9.84% (6/61)	9.84% (6/61)	11.48% (7/61)
Outlier dataset (all backdoor behaviour data)	58.23% (138/237)	76.79% (182/237)	77.22% (183/237)

Table 5.1: Outlier Detection Ratio.

metrics like Accuracy, Recall, Precision etc. rely on labels. The outlier detection ratio measures how many outliers a model detects in a given data set.

The first striking thing is that all three models show noticeably fewer outliers in the training data set than in the test data set. This deviation is not optimal and indicates underfitting. When calculating the underfitting metric in the next section, this can be confirmed. Next, you can see that IF detects outliers in data set “dataleak 1s” much worse than LOF and OCSVM. This also results in a worse outlier detection ratio on the “outlier dataset”, where all backdoor behaviour data sets are concatenated. All models perform poorly in detecting outliers in data set “dataleak 50s” dataset LOF and OCSVM are performance-wise very similar.

Jumping into the evaluation of the models towards trustworthiness, the methodology pillar can be analyzed once for all together. Since the methodology metrics do not focus on the model and its prediction of data, the methodology metrics have the same score output for all the three solutions. The metrics outcome for IF, LOF and OCSVM in the methodology pillar can be seen on figure 5.2. Further details on each metric are shown in figure 5.3.

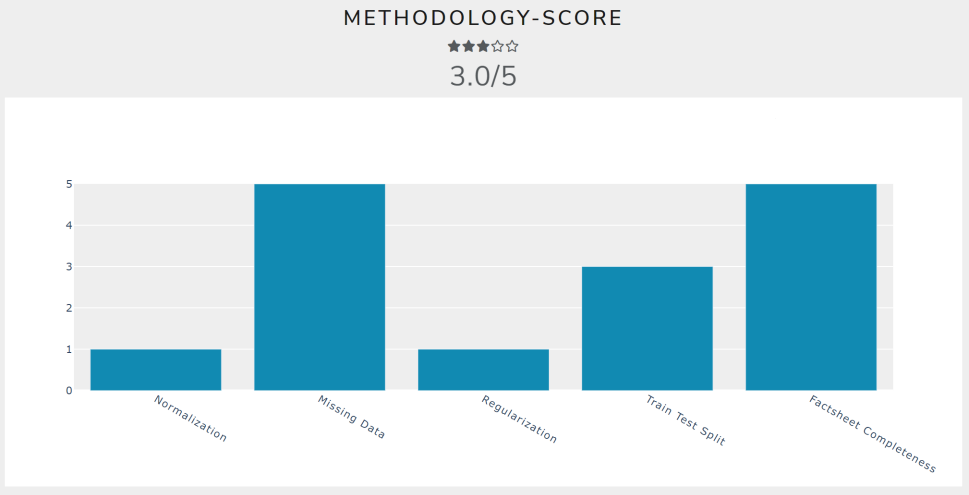

Figure 5.2: Analyze page: Methodology Score.

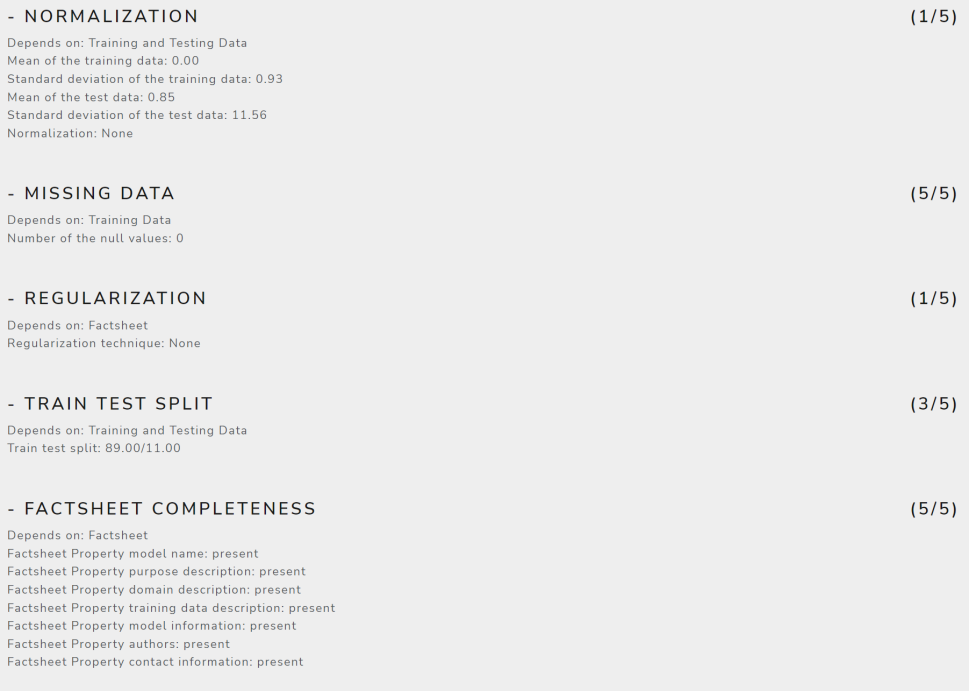

Figure 5.3: Analyze page: Methodology metrics details.

For the following solutions namely Isolation Forest, Local Outlier Factor and One-Class Support Vector Machine no protected features and protected values were defined since there is no privacy data to protect. Therefore, statistical parity difference and disparate impact are not computable and do not carry any weight in the final trust score.

5.3.1 Isolation Forest

In an Isolation Forest, data that has been sub-sampled randomly is processed in a tree structure using randomly chosen features. As they require more cuttings to separate, samples that travel further into the tree are less likely to be anomalies. Similar to this, samples that end up on shorter branches tend to be anomalies because the tree found it easier to distinguish them from other observations.

On figure 5.4 general information about the solution are shwon. The overall trust-score with the respective metrics are plotted in figure 5.5.

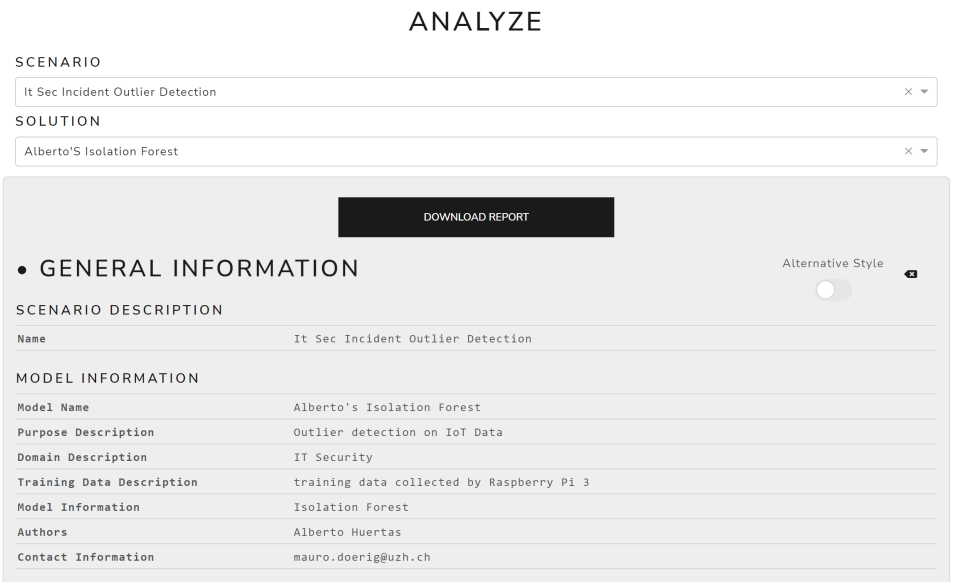

Figure 5.4: Analyze page: General information IF.

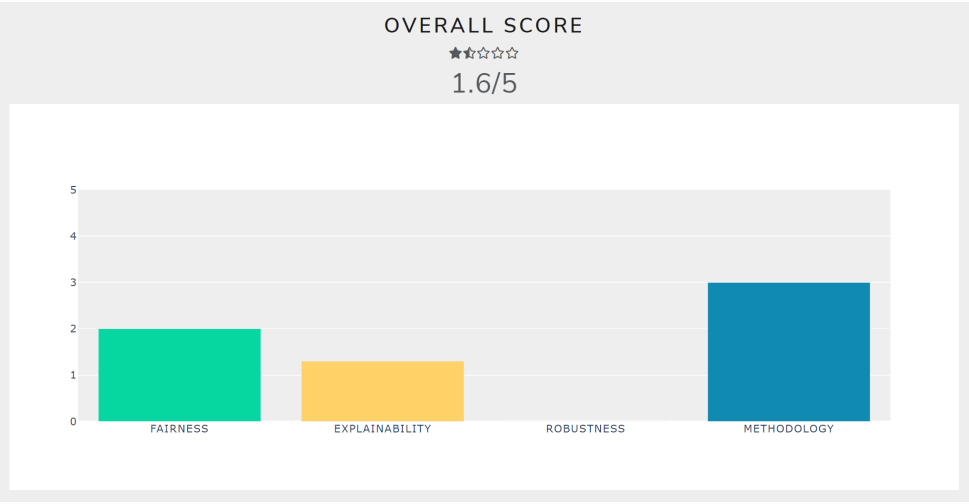

Figure 5.5: Analyze page: Overall trust score IF.

IF: Fairness Pillar

In the Fairness Pillar the only computed metric is underfitting. As seen in table 5.1 the training data and test data outlier detection ratio differ clearly which matches with the low underfitting score of 2.

Explainability Pillar

With a very high feature correlation of 42.65% computed by the correlated features metric, the score evaluates correspondingly low with a value of 1.

The low model size score of 2 is due to the large number of features, namely 67. A large number of features makes a model less explainable and thus less trustworthy. Since all three models have the same number of features, they all have the same model size score.

The last calculated metric in this pillar, the permutation feature importance metric, calculates 20 redundant features out of a total of 68 features. The high ratio of 29.41% redundant features results in a score of 1. The number of shuffling iterations is set to 8, which makes the computation more accurate. The three most important features are “kmem:kmem_cache_free”, “kmem:mm_page_free”, “writeback:writeback_dirty_inode” in descending order.

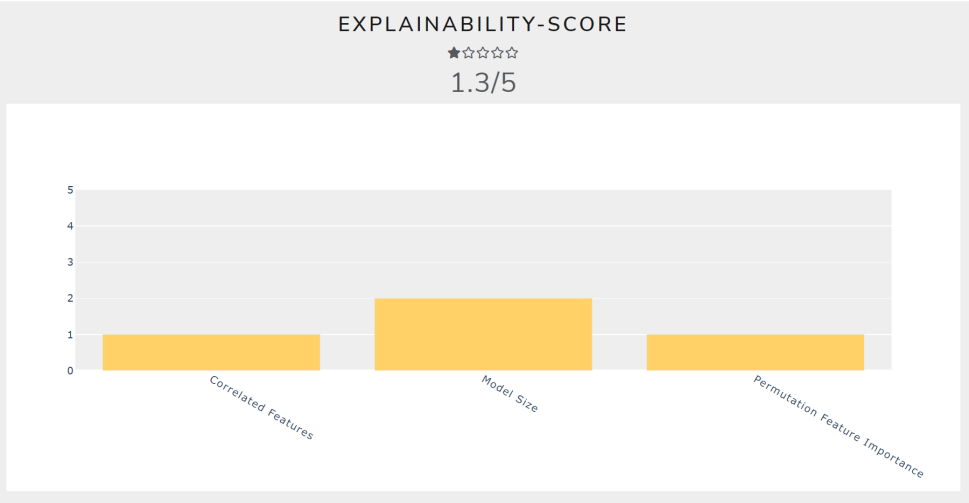

Figure 5.6: Analyze page: Explainability Score IF.

IF: Robustness Pillar

No Robustness metrics have been computed. The only applicable metric is the CLEVER Score metric which is only available for Keras models. Because this metric cannot be calculated, a very low trust score for the robustness pillar results, which in turn leads to a low overall trust score.

5.3.2 Local Outlier Factor

The Local Outlier Factor algorithm calculates the local density deviation of a particular data point with respect to its neighbors. It is an unsupervised anomaly detection technique. The samples that have a significantly lower density than their neighbors are considered outliers.

General information for the LOF solution is shown in figure 5.7.

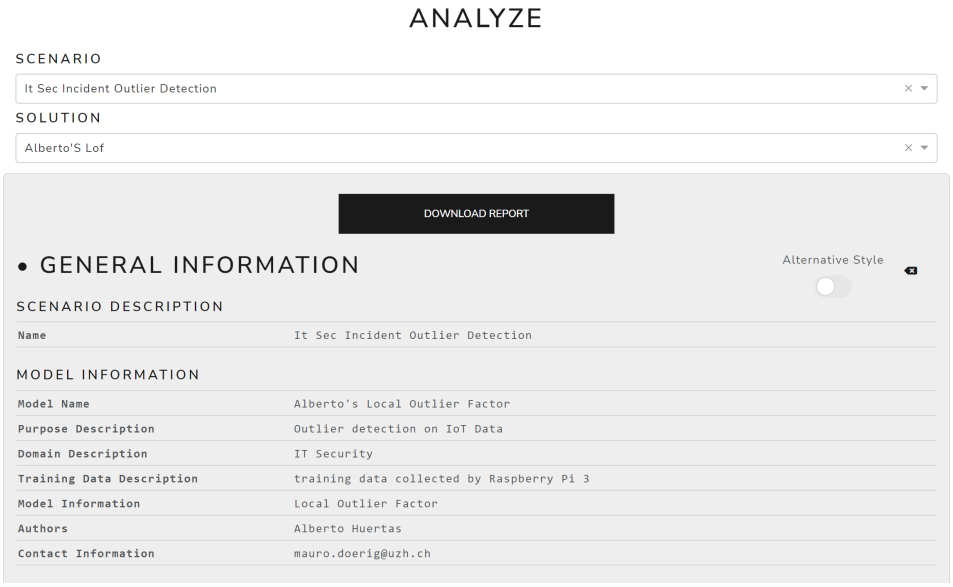

Figure 5.7: Analyze page: General information LOF.

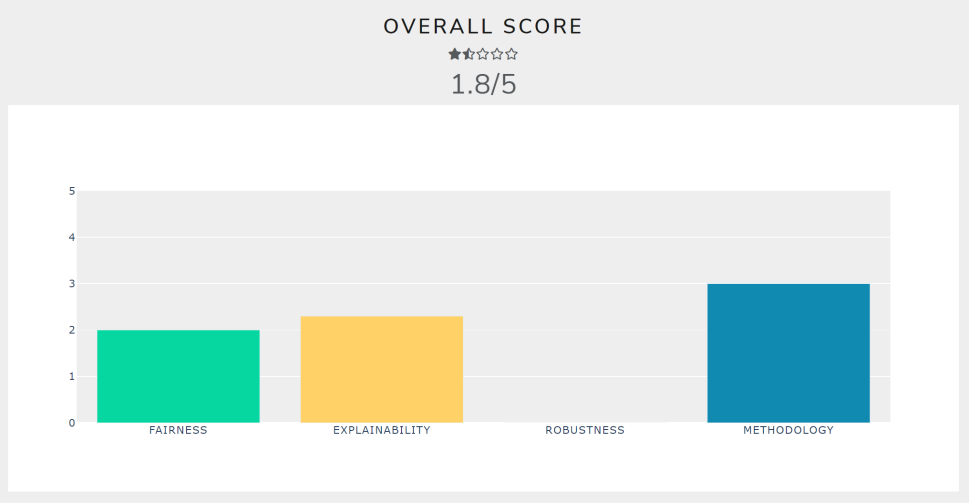

Figure 5.8: Analyze page: Overall Trust Score LOF.

LOF: Fairness Pillar

Similar to the Isolation Forest a large outlier detection ratio difference in the training and test data set exists which leads to a bad score of 2 in the underfitting metric.

LOF: Explainability Pillar

The metrics Correlated Features and Model Size have the same scores for all three models, since they both take only training and test data as input for its calculations, which are all the same for the three models.

The Permutation Feature Score performs better than Isolation Forest with only 4 redundant features out of a total of 68. this ratio of 5.88% redundant features results in a score of 4. “Tcp:tcp_probe”, “net:netif_rx”, “skb:consume_skb” are the most important features in descending order.

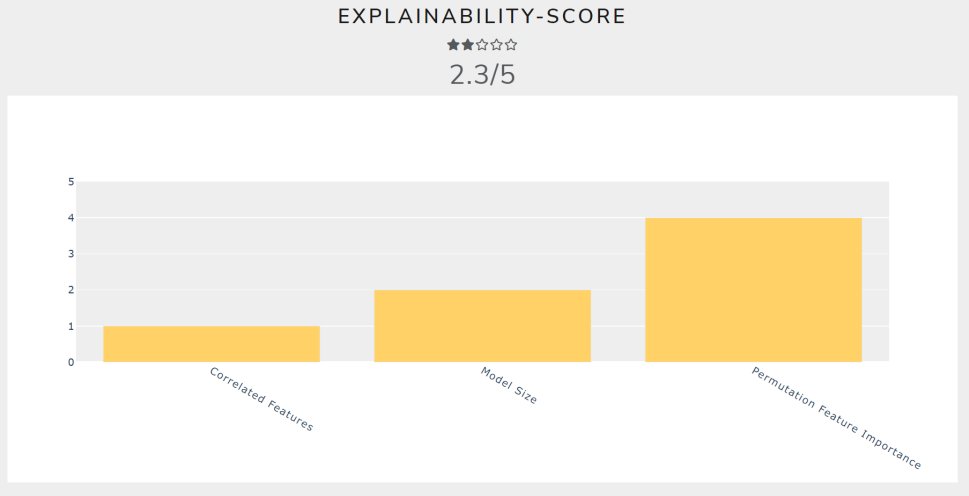

Figure 5.9: Analyze page: Explainability Score LOF.

LOF: Robustness Pillar

No Robustness metrics have been computed. The only applicable metric is the CLEVER Score metric which is only available for Keras models.

5.3.3 Learning One-Class Support Vector Machine

Contrary to the more popular approaches of binary classification or multi-class classification, where all classes are well described by the training data, the task of one-class classification is to characterize a single class that is well described by the training data and distinguish it from all others. The effectiveness of one-class support vector machine algorithms like OCSVM has been demonstrated in a variety of applications [4].

General information and overall trust score for OCSVM are shown in figure 5.10 and 5.11.

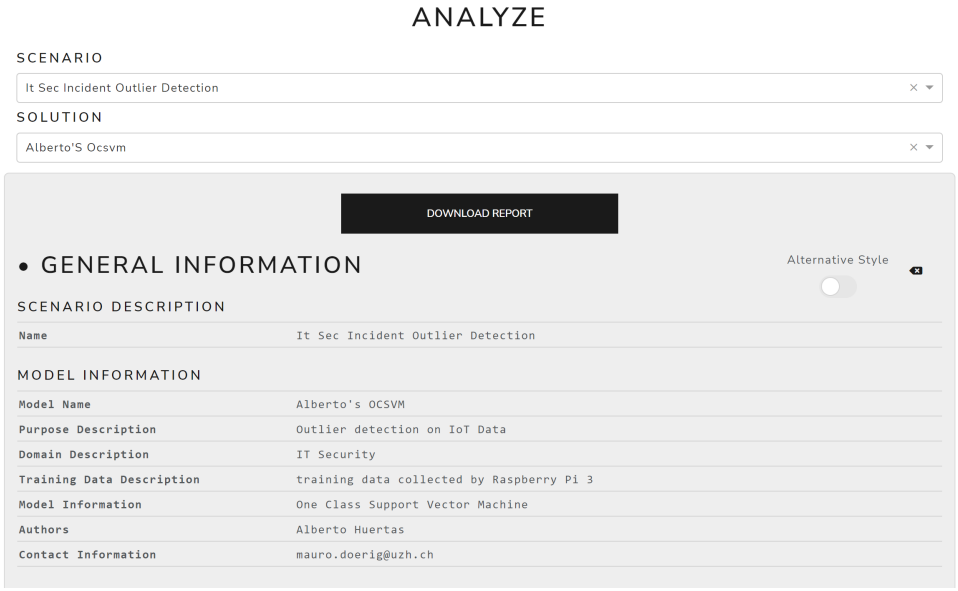

Figure 5.10: Analyze page: General information OCSVM.

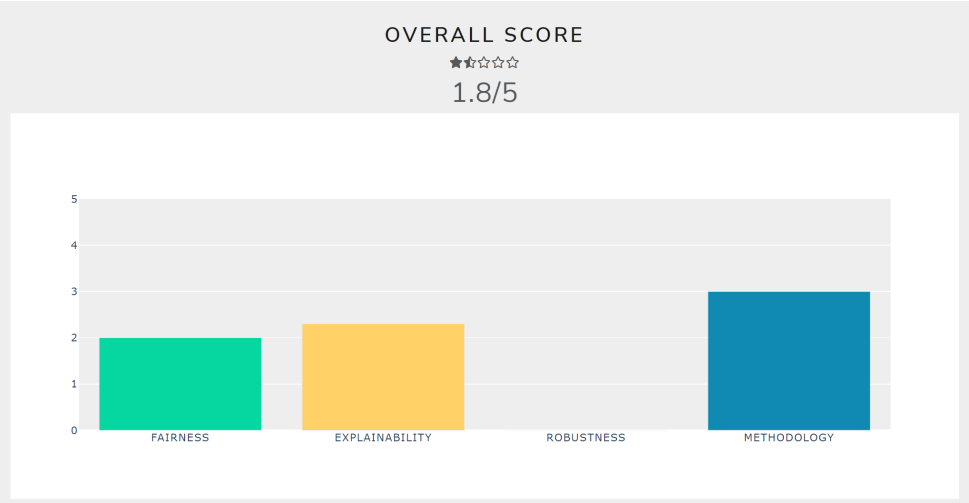

Figure 5.11: Analyze page: Overall trust score OCSVM.

OCSVM: Fairness Pillar

The underfitting metric computes a score of 2, which is because of the large outlier detection difference in the training and test data set. Similar as for IF and LOF. Values are shown in table 5.1.

OCSVM: Explainability Pillar

As for the correlated features and model size metric the outcomes evaluate to the same scores 1 and 2.

The permutation feature importance metric computes 6 redundant features out of 68 which gives a ratio of 8.82%. This ratio corresponds to a score of 4. The three most important features are “block:block_touch_buffer”, “writeback:writeback_dirty_inode_enqueue”, “signal:signal_deliver” in descending order.

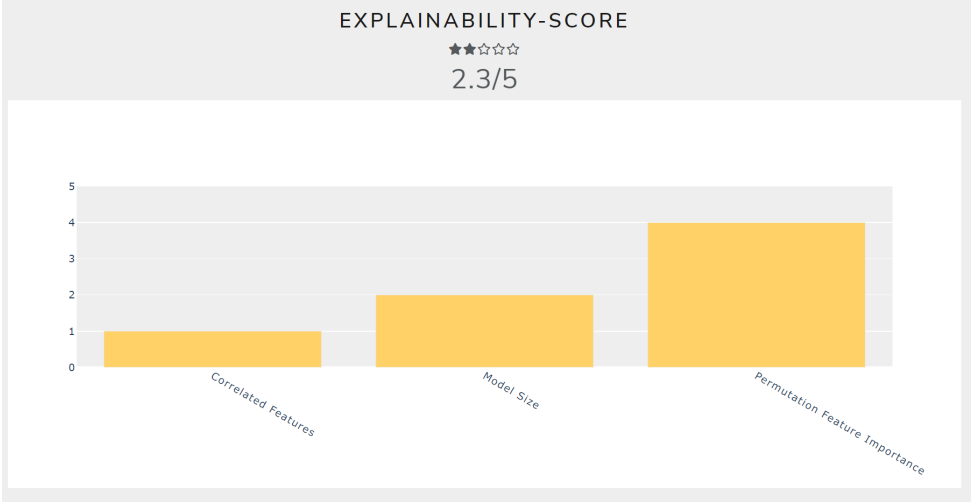

Figure 5.12: Analyze page: Explainability Score OCSVM.

OCSVM: Robustness Pillar

The CLEVER Score metric, which is exclusively available for Keras models has not been calculated.

Chapter 6

Discussion

6.1 Contribution

Taxonomy of most relevant metrics for unsupervised ML solutions

Literature has not yet proposed a general taxonomy of pillars and metrics to quantify the trustworthiness of unsupervised ML models. In the work of this bachelor thesis the main pillars of trust have been identified and aggregated to a total of four pillars (fairness, explainability, robustness, methodology). Also, the most relevant metrics for each pillar have been reviewed and analysed for the application in unsupervised ML environment.

Adopting supervised metrics into unsupervised metrics

Most of the metrics to quantify the level of trustworthiness have been proposed for supervised ML models. The main reason is that they rely on labels for its computation. Another reason is that the ground-truth without labels is not given in unsupervised ML environment and makes some metrics outcome meaningless. As a solution to this problem the outlier data set was taken to calculate specific metrics. A disadvantage of this approach is that it serves not as a general solution for all unsupervised ML solutions, but specifically for unsupervised anomaly detection solutions. By having an implicitly labeled outlier data set at hand where only outliers occur, it becomes possible to transform supervised metrics into unsupervised metrics.

A lot of effort has been put into transforming metrics originally proposed for supervised solutions into metrics for unsupervised anomaly detection solutions. This approach has not yet been proposed in literature and is the main contribution of this bachelor thesis.

Algorithm to compute trustworthiness for unsupervised anomaly detection models

As described in chapter 4 Trusted-AI is a proposed algorithm to quantify the trustworthiness for supervised ML models. The idea of computing each pillar's metrics individually and aggregating the weighted scores to a final trust score was adopted for unsupervised ML models.

The main work for this part of the bachelor thesis consisted of integrating the algorithm for computing the trustworthiness of unsupervised anomaly detection models into the existing Trusted-AI algorithm. Since the metrics for unsupervised anomaly detection models differ from the supervised metrics they had to be newly implemented. Also mappings and weights configurations were adjusted to fit the unsupervised environment.

Extension of trusted-AI Webapp for unsupervised ML solutions

The heavily entangled application did not permit a clear separation at the top level of code, so each page was processed separately. Some parts of the application could be reused, others did not exist yet. The folder structure in the codebase is customized for each page and is divided into supervised and unsupervised categories. This restructuring now also allows to better extend the application in case semi-supervised scenarions or others are added.

In the frontend, the separation between supervised and unsupervised becomes apparent as soon as the toggle is switched. If the unsupervised mode is on, the application in the background branches to the unsupervised part for the metric calculations.

Validation of the newly implemented algorithm for unsupervised anomaly detection solutions

The implemented algorithm for unsupervised solutions was tested using one application scenario, which verified the algorithm's usability and gave in-depth insights into the reliability of the various solutions with visualizations and comprehensive explanations of each metric. The application allows the user to fully customize the parameters to any application case.

6.2 Limitations

The work of this bachelor thesis does not present a general solution for all unsupervised scenario solutions to quantify their trustworthiness, but is limited to unsupervised anomaly detection scenarios where an implicitly labeled data set can be given as input (in this case outlier data set). However, there are still several metrics presented that can be computed without the need of such a data set. Most of these metrics are contained in the methodology pillar.

Another limitation is presented in the robustness pillar, where most of the metrics only exist for supervised ML models. The literature review has not yet proposed such metrics and there is a lack of scientific work in this area. The only applied metric in the Trusted-Anomaly-Detection algorithm was the CLEVER metric that was originally proposed for supervised ML models and was adopted for unsupervised ML models within the scope of this thesis. For other supervised metrics related to robustness it was not possible to adapt it in the same style, because they relied on labels for its computations.

Chapter 7

Summary and Conclusions

In the first part of this bachelor thesis, an in-depth literature review was conducted focusing first on supervised metrics and then on unsupervised metrics related to trustworthiness. A deeper understanding of different trust pillars was established also by analyzing the work of my predecessors on Trusted-AI ([12]). The literature research became increasingly more specific to unsupervised anomaly detection, with first a global search for all possible existing metrics and then for each pillar individually. This was one of my main challenges, as even after intensive literature research, little could be found in this area.

When i moved from literature research to practical implementation in the next step of this work, I first concentrated on programming prototypes of the metrics on a separate jupyter notebook. The main focus was to analyze and reprogram already implemented metrics from Trusted-AI so that they could be used for unsupervised anomaly detection solutions.

Parallel to this work, there was a constant search in the scientific literature for the same or similar work to confirm and support my work. In the literature, as already mentioned before, there was little to almost no comparable work and the necessity for such work was underlined.

The programmed prototypes of the metrics were looked at with my supervisors Dr. Alberto Huertas and Muriel Franco. Some of them have had several iterations of improvement. After this work was completed, it was time for the final implementation of the Trusted-Anomaly-Detection algorithm with all its pillars and metrics to extend the existing webapp. Extending Trusted-AI presented some difficulties, firstly it took a lot of time to understand the complex code base and to learn the framework used. After that a method was worked out to extend the code base in the best possible way, which consisted of separating each page separately for a supervised and unsupervised approach. Each page in turn presented its own problems, as components were so different from each other that much of the code had to be rewritten from scratch.

Finally, several trained models were used to evaluate Trusted-Anomaly-Detection. It has been shown in chapter 5 that Trusted-Anomaly-Detection algorithm is able to quantify the trustworthiness of unsupervised anomaly detection models. This project helps in bridging the knowledge and trust gap between model consumers and model developers.

7.1 Future Work

7.1.1 Improvement of Computation Speed

Computing all metrics takes a rather high amount of time. The metric Permutation Feature Importance is the most significant, since each feature column has to be shuffled for a given number of times. By setting down the number of shuffles the computation time decreases on one side, but on the other side the computed importance of a feature becomes less meaningful. A trade-off of 3 shuffling periods has been chosen.

For future work more sophisticated shuffling algorithm could be explored to bring down the computation time.

7.1.2 Additional Robustness Metrics

For future work, additional robustness metrics should be evaluated and added. Currently, the entire robustness pillar trust score depends on a single metric, which means that no meaningful statement can be made about this pillar. In this bachelor thesis, a lot of research was done to find additional metrics, but it turned out to be a big challenge because not much could be found on this topic. With even more in-depth research, this gap could possibly be filled in future work.

7.1.3 Webapp Usability

If a user wants to analyze a solution to a scenario the computation is triggered in the backend, but does not give any feedback. This lack of usability could be fixed by implementing a loading spinner to give a user better feedback of what is going on in the background.

7.1.4 Extension for Other Unsupervised Scenarios or Semi-Supervised Scenarios

This work has focused on unsupervised anomaly detection scenarios. The webapp however does not cover all unsupervised scenarios to compute the trustworthiness. Also, semi-supervised scenarios are not covered. Extending the webapp for other unsupervised scenarios and semi-supervised scenarios could be part of future work. The major restructuring conducted in this bachelor thesis supports the extension for other scenarios to a higher degree than before.

Bibliography

[1] The 21st century's ai biggest achievements.

[2] Amazon scraps secret ai recruiting tool that showed bias against women howpublished = https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-iduskcn1mk08g, note = Accessed: 2022-25-08 author = Jeffrey Dastin.

[3] M Arnold et al. Factsheets: increasing trust in ai services through supplier's declarations of conformity. corr (2019). *arXiv preprint arXiv:1808.07261*.

[4] Abdenour Bounsiar and Michael G. Madden. One-class support vector machines revisited. In *2014 International Conference on Information Science & Applications (ICISA)*, pages 1–4, 2014. doi: 10.1109/ICISA.2014.6847442.

[5] A Rogier T Donders, Geert JMG Van Der Heijden, Theo Stijnen, and Karel GM Moons. A gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10):1087–1091, 2006.

[6] Fairlearn. Improve fairness of ai systems. https://github.com/fairlearn/fairlearn, 2022.

[7] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20(177):1–81, 2019.

[8] Alex A. Freitas. Comprehensible classification models: A position paper. *SIGKDD Explor. Newsl.*, 15(1):1–10, mar 2014. ISSN 1931-0145. doi: 10.1145/2594473.2594475. URL https://doi.org/10.1145/2594473.2594475.

[9] IBM. adversarial-robustness-toolbox. https://github.com/Trusted-AI/adversarial-robustness-toolbox, 2022.

[10] IBM. Ai explainability 360. https://aix360.mybluemix.net/?_ga=2.88374680.2057431906.1637135405-1405718511.1620651272, 2022.

[11] IBM. Ai fairness 360. https://github.com/Trusted-AI/AIF360, 2022.

[12] Melike Demirici Jan Bauer, Joel Leupp. Quantifying the Trustworthiness Level of Artificial Intelligence Models and Decisions. Master's thesis, University of Zurich, Department of Informatics, 2022.

[13] T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1):1793–8201, 2011.

[14] Sevvandi Kandanaarachchi, Mario A Muñoz, Rob J Hyndman, and Kate Smith-Miles. On normalization and algorithm selection for unsupervised outlier detection. *Data Mining and Knowledge Discovery*, 34(2):309–354, 2020.

[15] Will Koehrsen. Overfitting vs. underfitting: A complete example. *Towards Data Science*, 2018.

[16] Christoph Molnar. Permutation feature importance. `https://christophm.github.io/interpretable-ml-book/feature-importance.html`, 03 2022. Accessed: 2022-06-28.

[17] Minh-Nghia Nguyen and Ngo Anh Vien. Scalable and interpretable one-class svms with deep learning and random fourier features. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 157–172. Springer, 2018.

[18] D Nkashama, Arian Soltani, Jean-Charles Verdier, Marc Frappier, Pierre-Marting Tardif, and Froduald Kabanza. Robustness evaluation of deep unsupervised learning algorithms for intrusion detection systems. *arXiv preprint arXiv:2207.03576*, 2022.

[19] Cynthia Rudin, Caroline Wang, and Beau Coker. The age of secrecy and unfairness in recidivism prediction. *arXiv preprint arXiv:1811.00731*, 2018.

[20] Yingjie Tian and Yuqi Zhang. A comprehensive survey on regularization strategies in machine learning. *Information Fusion*, 80:146–166, 2022.

[21] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[22] Chathurika S. Wickramasinghe, Kasun Amarasinghe, Daniel L. Marino, Craig Rieger, and Milos Manic. Explainable unsupervised machine learning for cyber-physical systems. *IEEE Access*, 9:131824–131843, 2021. doi: 10.1109/ACCESS.2021.3112397.

[23] Shuhan Yuan and Xintao Wu. Trustworthy anomaly detection: A survey. *arXiv preprint arXiv:2202.07787*, 2022.

[24] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th international conference on world wide web*, pages 1171–1180, 2017.

[25] Hongjing Zhang and Ian Davidson. Towards fair deep anomaly detection. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 138–148, 2021.

List of Figures

3.1 Intuition behind CLEVER [21]. 19

4.1 Trusted-AI [12]: Taxonomy. 24

4.2 Trusted-AI: Final trust score computation 27

4.3 Webapp: Architecture 28

4.4 Webapp: Original Navigation Bar. 28

4.5 Webapp: Navigation Bar (in supervised mode). 29

4.6 Webapp: Navigation Bar (in unsupervised mode). 29

4.7 Scenarios page: Supervised scenarios. 30

4.8 Scenarios page: Unsupervised scenarios 31

4.9 Scenarios page: Create new unsupervised scenario. 31

4.10 Scenarios page: New unsupervised scenario. 32

4.11 Upload page: Upload supervised solution. 33

4.12 Upload page: Upload unsupervised solution. 34

4.13 Analyze page: Select unsupervised scenario and solution. 35

4.14 Analyze page: General information. 35

4.15 Analyze page: Trustworthiness report. 36

4.16 Analyze page: Methodology metrics scores. 37

4.17 Analyze page: Methodology metrics details. 37

4.18 Compare page: Select and compare two solutions for a scenario. 38

4.19 Compare page: Comparison of two selected solutions. 39

5.1 Analyze page: Properties. 42
5.2 Analyze page: Methodology Score. 44
5.3 Analyze page: Methodology metrics details. 44
5.4 Analyze page: General information IF. 45
5.5 Analyze page: Overall trust score IF. 45
5.6 Analyze page: Explainability Score IF. 46
5.7 Analyze page: General information LOF. 47
5.8 Analyze page: Overall Trust Score LOF. 47
5.9 Analyze page: Explainability Score LOF. 48
5.10 Analyze page: General information OCSVM. 49
5.11 Analyze page: Overall trust score OCSVM. 49
5.12 Analyze page: Explainability Score OCSVM. 50

List of Tables

3.1 Underfitting Score Map. 9

3.2 Overfitting Score Map. 10

3.3 Statistical Parity Difference (SPD) Score Map. 12

3.4 Disparate Impact (DI) Score Map. 13

3.5 Overview Fairness Metrics 13

3.6 Feature Correlation Score Map. 15

3.7 Model Size Score Map. 16

3.8 Permutation Feature Importance Score Map 17

3.9 Overview Explainability Metrics. 18

3.10 Overview Robustness Metrics. 19

3.11 Overview Methodology Metrics. 20

3.12 Score Map Regularization. 20

3.13 Missing Data Score Map. 21

3.14 Normalization Score Map. 22

4.1 Metrics for unsupervised solutions. 25

5.1 Outlier Detection Ratio. 43