University of Zurich[UZH]

MASTER PROJECT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Quantifying the Trustworthiness Level of Artificial Intelligence Models and Decisions

Joel Leupp
Melike Demirci
Jan Bauer
Zurich, Switzerland

Supervisors: Dr. Alberto Huertas & Muriel Franco
Date of Submission: February 15, 2022

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

ifi

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

Abstract

The ever increasing use of Artificial Intelligence (AI) in high-stake areas, constantly increases the required level of trustworthiness and reliability of such systems. The broad adoption of AI systems will rely on the ability to trust their decisions. In order to trust automated systems, it is important to understand their reasoning, to analyse if there is a hidden bias involved and to estimate how robust the predictions are. IBM's "Watson for Oncology" system, for example failed to deliver on its promise to generate personalized treatment recommendations for cancer patients, by producing advice that is "unsafe and incorrect". This spikes the need for automatic tools, that can be used to calculate and communicate trust levels of AI. The algorithm being proposed, evaluates the model with regard to the main pillars of trust - fairness, explainabilty, robustness and training methodology. For every pillar a set of, state of the art, metrics was compiled into a unified taxonomy. The trust certification algorithm was implemented as a graphical python application. The developer uploads the data used, trained model and factsheet (containing the most important meta data) and then receives an analysis of how trustworthy the model is estimated to be. The application also offers the possibility to compare two models with each other. At the end of the process, a report containing the full analysis, including metrics and their assessment, can be downloaded. This report helps to establish and communicate a models trustworthiness among the various stakeholders. In order to validate the algorithm implementation, a real world scenario, focused on the cybersecurity of IoT devices, was used. Various machine learning models were trained and helped to test and refine our algorithm.

Zusammenfassung

Der immer häufigere Einsatz von Künstlicher Intelligenz (KI) in kritischen Bereichen, wie zum Beispiel dem Gesundheitswesen, erhöht ständig die Anforderungen an die Vertrauenswürdigkeit und Zuverlässigkeit solcher Systeme. Die breite Akzeptanz von KI hängt maßgeblich davon ab, ob man sich auf deren Vorhersagen verlassen kann. Vertrauen in automatisierte Systeme kann gewonnen werden, indem man die Entscheidungsfindung nachvollzieht, analysiert ob versteckter Bias existier und abschätzt wie robust die Vorhersagen sind. IBM's "Watson for Oncology", welches personalisierte Behandlungen für Krebspatienten vorschlagen sollte, konnte die gesteckten Erwartungen nicht erfüllen, da es unsichere und falsche Vorschläge machte. Dies zeigt, wie wichtig Mechanismen und Methoden sind, mit denen man automatisch und zuverlässlich berechnen kann, wie vertrauenswürdig ein KI Modell ist. Unser Beitrag besteht in der Entwicklung eines Algoritmus der diesen Zweck erfüllt. Um ein Modell hinsichtlich Fairness, Erklärbarkeit, Robustheit und Trainingsmethodik zu bewerten, werden eine Reihe von Metriken berechnet und interpretiert. Der Algorithmus wurde prototypisch in einer Python-basierten Webanwendung umgesetzt. Der Entwickler lädt die Daten, das trainierte Modell und das Factsheet (mit den wichtigsten Metadaten) hoch und erhält im Anschluss eine Analyse, wie vertrauenswürdig das Modell ist. Die Anwendung bietet auch die Möglichkeit, zwei Modelle miteinander zu vergleichen. Für jedes Model lässt sich dadurch ein Report erstellen, der eine vollständigen Analyse, inklusive Metriken und deren Bewertung enthält. Dieser Report kann dabei helfen, Vertrauen unter den verschiedenen Anwendern zu etablieren. Um die Implementierung des Algorithmus zu validieren, wurde ein reales Szenario, das sich auf die Cybersicherheit von IoT-Geräten konzentriert, verwendet. Verschiedene Modelle wurden trainiert und benutzt um den Algorithmus zu testen und zu verbessern.

Acknowledgments

First of all, we would like to thank all members of the Communication Systems Research Group at UZH for supporting us throughout the project. Many thanks to Dr. Muriel Franco and Prof. Dr. Burkhard Stiller for recommending us to the right place and making it possible for us to work on this project. Our sincere thanks go to our supervisor Dr. Alberto Huertas for guiding the project and for always assisting us with words and deeds. Eder Scheid, who has helped us with the deployment of our application has to be mentioned here as well.

A special thanks also goes to IBM for their work on trust in AI, that inspired and serves as the basis of our project. We are indebted to Kush Varshney for providing us with his wealth of knowledge in the field.

We are happy that our project has been completed after almost a year of active work and we are proud of what we have achieved. We are grateful for our families and partners unconditional support during this challenging time.

Contents

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) made great strides over the last decade. In 2011 IBM Watson was able to beat champions in the open-domain Q&A game of Jeopardy for the first time. In 2016, AlphaGo defeated world champion Lee Sedol in the game of Go [74]. Such events are seen as major breakthroughs. With increasing capabilities, AI gained more relevance as support to human decision making, spanning trivial to highly complex applications. For example, medical applications like the diagnosis and treatment of diseases, over the assessment of legal issues, employment decisions, college admission to credit scoring. In most scenarios, the machine does not act completely autonomously and rather supports human decision making. This support enables humans to take better overall decisions. However, in complex cyber-physical systems, like autonomous driving or robotic surgery, some decisions and actions have to be taken autonomously. AI is also increasingly used in IT security, to automatically detect anomalies like cyber-attacks and intrusions.

The act of delegating parts of the decision making process to an automated and intelligent system generates a trust dependency. Being able to trust an AI-based model is particularly important in high-stakes situations. For example, a software, supporting judges in pretrial recidivism scoring was shown to be racially biased [67]. Similarly, a model used for applicant screening at a big technology company discriminated against women by perpetuating existing bias in the training data. Past fatalities connected to autonomous vehicles often happened in highly unlikely situations for which the underlying model was not prepared nor trained for [82]. Stories of situations in which a wrong decision had far-reaching negative consequences erode the trustworthiness of such systems. When humans rely on other humans to make decisions for them, they engage in dynamic trust relationships with each other. Human trust can be established through empirical evidence, demonstration and explanation. A surgeon, who has already performed the respective surgery 1000 times is trusted more than one, who does it for the first time. Academic titles, certifications, and positive recommendations can provide the basis of trust. Transactions involving significant risk would not be possible without trust. Trust is also the central component of the interaction between people and AI and its acceptance. The literature has identified explainability, robustness, fairness, and methodology as key aspects of trust in AI.

For external stakeholders, it is often completely impossible to understand the inner workings of the system nor how an AI program is making its predictions [13]. With increasing model complexity and impact, trust quickly diminishes. For human action and decision-making, trust can be gained through verbalized explanation of the underlying rationale. On the contrary, for deep learning models with millions of parameters, it is almost impossible to understand how the decision is taken. Explainability in ML is concerned with understanding how a model came to its conclusion. For the use of machine learning models in high-risked fields, it is important that all stakeholders can understand the main decision drivers [21].

Considering fairness, bias is one of the main issues against trusting an AI system. Bias can be introduced by prejudice in the training data or assumptions made during the training process. ML models inherit human biases which were included in the training data. In some cases, the collected dataset might be too small, incomplete or lack diversity. Then, it would be not representative of the underlying population. In order to improve trust in an AI system, unjustified bias must be removed, in order to enable fair algorithmic decision making [69].

In the case of robustness, when a machine learning model is created and deployed in the physical world, it is impossible to control every single input that is given to the system. AI systems can be targets for attacks by malicious actors. Even in the black-box setting where the parameters of the model are unknown to the attacker, it is possible to create unexpected and dangerous outcomes using adversarial examples. The output of the machine learning algorithms must be stable and robust, even against adversarially created inputs. Being able to show and prove that a model is secure and invulnerable against attacks creates trust [5].

In terms of methodology, knowing more about the applied training and the evaluation process increase trust. For this purpose, the approach that is taken for the construction of an AI system should be transparently communicated to all stakeholders. The decisions that are taken by the developer of the AI system, how it was built and how it was evaluated provide valuable insights about the system's strengths and potential weaknesses.

Trust in AI is an emerging research topic which needs to be addressed carefully. The requirements for trust in AI will be further discussed and clarified and methods to quantify trustworthiness still need to be fully developed. So far the state of the art is still lacking a complete collection of metrics for each pillar of trust. A methodology that can be used to aggregate the set of evaluated metrics into a final trust score will be proposed. Furthermore, another important requirement is to validate the proposed solution using a real world application scenario.

1.2 Description of Work

To improve the previous challenges, this work combines the four pillars of trust and the applicable metrics into a general taxonomy. Each metric takes the data, the model, and the associated factsheet as input. The taxonomy also mentions for every metric what preconditions have to be met, for the metric to be computable. A general taxonomy of trust in AI, combining different pillars and associated metrics together, has not yet been created.

From the general taxonomy an algorithm, able to quantify the trustworthiness level of trained ML models, is derived. The algorithm first calculates all available metrics for each pillar, weights them according to user preferences, and aggregates them into a pillar score. The individual pillar scores are then weighted and aggregated into a final trust score. According to the application scenario, the user might assign individual weights to every metric and pillar. For example, in the case of policy analysis, users might prioritize explainability and fairness over robustness and methodology by assigning higher weights to the former two. When calculating the final trust score, these two pillars will then have a stronger influence.

The algorithm has been implemented as a proof-of-concept in a Python-based web application, available on a university server. The web application provides an interface for users, where ML models and the data that the model was trained on can be uploaded. This triggers the trust score calculation, graphically visualizing how the model scored concerning every metric. The results are compiled into a report, able to provide deep insights regarding the model and its trustworthiness. In order to identify and inspect the differences, two models and their scores can be compared side by side.

The web application and the trusted AI algorithm were validated on two different application scenarios. The first one is an IT security scenario where the goal is the classification of different types of attacks on IoT devices. The second one is a credit card approval scenario for a financial institution. Various models with different hyperparameters and configurations were trained and then used to evaluate the algorithm.

1.3 Thesis outline

This thesis comprises four main chapters. Chapter 1 is intended to provide the reader with a motivating introduction to the topic and explain how the challenges presented were approached.

In Chapter 2, a detailed analysis of the central pillars of trust is carried out. The definitions and specific aspects of fairness, explainability, robustness and methodology are listed and explained. For each pillar, the most important metrics used to carry out a model assessment, are listed. Moreover, the limitations of each pillar and metric are discussed to provide a complete overview.

Chapter 3 reviews existing related theoretical and practical work. Comparable methodologies and existing tools are taken into consideration and compared to this work. The related work is put into perspective, to evaluate how synergies can be used and created.

Chapter 4 contains the main contributions of this work. It contains the general taxonomy of pillars and metrics for trust in AI and the functional details of the trusted AI algorithm are explained. It also explains how the Trusted AI Algorithm was implemented and made available as a Python-based web application. Two different application scenarios are used in order to validate and evaluate the proposed solution. For each scenario, multiple models were trained and helped to assess and improve the algorithm and its implementation.

Finally, Chapter 6 concludes this work by providing a summary of outcomes. Possible limitations are discussed and act as a motivation for future work on trust in AI.

Chapter 2

The Pillars of Trust in AI

Statistical models trained through Machine Learning and Deep Learning, are increasingly becoming key pieces of products and services. The deployment of these systems in high-stakes areas, such as credit applications, judicial decisions, and medical recommendations, requires greater trust [11]. However, a survey amongst business leaders has shown that trusting AI systems is very challenging for them. As humans, we frequently rely on automated systems. For example, we accept that the autopilot will operate an airplane, trusting that it will navigate correctly. According to IBM, trust in AI is placed on four key pillars - namely fairness, explainability, robustness and methodology. To ensure trust in AI-based outcomes, a transparent AI lifecycle is needed. At every step of the lifecycle, the most important design decisions need to be documented and transparently communicated to the various stakeholders.

Despite active research and development to address these issues, there is no mechanism yet for the creator of an AI service to communicate how they are addressed in a deployed version [11]. A so called model "Factsheet" will contain sections on all relevant attributes of an AI service, such as intended use, performance, potential bias, safety, and security information. The factsheet should contain the most important information regarding a model, for example, for which use case the model was created and on which data it was trained and tested. The most important performance indicators should also be included. These Factsheets are modeled after defined standards and can fulfill a similar purpose to approaches from other industries, like energy star or food nutrition labels [11].

In the following sections, the four key pillars of trust in AI are introduced and discussed. For every pillar, it is motivated and explained why it is considered a key element and what metrics can be used to assess it. Each subsection concludes with a discussion of the limitations of each pillar.

2.1 Fairness Pillar

2.1.1 Motivation

At first, the importance of fairness in the context of AI is motivated. Then, metrics are described that can help to account for different aspects of fairness. Finally, existing limitations are discussed. Since, an increasing number of information systems make decisions based on statistical inference rules, acquired through techniques of machine learning and most recently deep learning. Especially in decision-critical contexts, such as predictive policing [65], credit scoring [50], and the allocation of health care resources [59], fairness is a top priority. Due to the fact that no singular nor general definition of fairness exists, to avoid encoding discrimination in AI-based systems, multiple fairness aspects need to be accounted for [58]. Fairness became increasingly important in recent years as machine learning conquered more application domains [76]. A system or decision is usually referred to as fair if it is impartial or neutral with regard to certain protected aspects. However, the specific definition of fairness depends on the defining subject and the scenario.

Legislative rules exist in order to protect people from discrimination based on *sensitive* or *protected attributes* like religion, ethnicity or demography [86]. For example, the American Fair Housing Act of 1968 prohibits discrimination in housing because of race, color, national origin, religion, sex, familial status and disability [40]. A *protected attribute* partitions the population into subgroups who should receive equal treatment, regardless of an individual group membership [11]. Besides humans, automated systems can also exhibit discriminatory behaviour in their decision making process. As a consequence, individuals or whole groups might be treated disparately. In order to recognize and prevent discrimination, fairness concerns regarding automated systems are rising. Several recent studies revealed cases of discrimination by AI [62]. For example, search ads for highly paid positions were less likely to be presented to women [22]. Critics voiced concerns regarding bias and discrimination in decision systems that rely on statistical inference learning [87].

Depending on the context, an AI-based model can be classified as unfair for a variety of reasons. First of all, the model may have been created for an unfair purpose like predicting race, gender, sexual orientation or applications like racial profiling, mass surveillance, or Internet censorship [51][54]. Since the decision making process is purely based on mathematical equations, it has no inherent moral understanding. Therefore, AI does not distinguish between ethical and unethical decisions. The human has to answer this type of questions, based on the context. For example, it is considered fair, if a system uses the gender of a patient for an individual treatment recommendation, if men and women are different enough with regard to the necessary treatment. On the other hand, it would be considered unfair to use the applicants' gender for a hiring decision. In order to ensure fairness, it is important to check the data and model for hidden bias [25].

Bias - A necessary Evil?

ML and AI are far from objective and impartial, making them prone to systematic error. So called bias, might place privileged groups at a systematic advantage over unprivileged

groups. Unwanted bias can be introduced into the final model during every step of the life-cycle. The dataset of collected samples that is used to train the model should be closest to the underlying population and represent all nuances of the target population. If people chose to volunteer for a study they might share characteristics making them different from non-participants (self selection bias) [45]. If poor randomization is used a selection bias is introduced into the dataset. Sampling bias emerges if the sample is collected in such a way that certain members of the intended population have a lower or higher sampling probability than others. This happens for example if a certain subset of data is excluded (exclusion bias) [79]. Overall, bias refers to systematic distortions of the results due to incorrect assumptions, making the sample unrepresentative of the population intended to be analyzed. Humans are very susceptible to bias and already over 180 human biases got identified and described [69]. A dataset representing human decisions mirrors at least one human bias. Such as a judge who gives stricter penalties to colored people than to white people. A model trained on this data, would learn and continue to discriminate against people of colour.

> "When the data we feed the machines reflects the history of our own unequal society, we are in effect asking the program to learn our own biases." [17]
>
> - ***Stephen Buranyi***

Many examples exist, where bias propagated from the training data into the model. Some courts, for example, use risk assessments models in order to decide who can be set free at every stage of the criminal justice process. If these models are trained on biased data, they might end up discriminating against defendants based on ethnicity. Angwin et al. showed that the risk scores of white defendants were heavily skewed towards low risk in contrast to the risk scores of people of color [10]. Apple's credit scoring algorithm was investigated after discriminating against women [29]. Feedback loops in the deployed system can also reinforce the future prediction by, for example, pushing a certain type of people out of the system. Image searches for professions such as CEO or software developer produce fewer images of women. Conversely, this might then lead to fewer women embarking on one of these positions. In order to mitigate bias different bias mitigation algorithms can be used for pre-processing, in-processing and post-processing. These, however, exceed the scope of our project.

However, without assumptions, an algorithm would have no better performance on a task than if the result was chosen at random, a principle which was formalized by Wolpert in 1996 into what we call the No Free Lunch theorem. According to the "No Free Lunch theorem" (Macready, 1997), all classifiers have the same error rate when averaged over all possible data generating distributions. Therefore, a certain classifier must have a certain bias towards certain distributions and functions to be better at modeling those distributions, which would at the same time render it worse at modeling other types of distributions.

Fairness through unawareness

Fairness through unawareness means that the prediction rule of a model does not consider protected features. For data points $X = \{x_1, x_2, \ldots, x_N\}$ and labels $Y = \{y_1, y_2, \ldots, y_N\}$ and protected feature P, instead of using all features, only the unprotected features X are used for training. This is supposed to ensure independence from the protected features. However, the naive approach of "fairness through unawareness" is ineffective due to the existence of redundant encoding. A redundant encoding exists if a protected attribute is highly correlated with an unprotected one. It was shown that in America the ZIP-Code of a person is strongly correlated with their ethnicity. Redlining, for example, denied people from districts with a high default risk access to home loans. The decision was taken based on the peoples' ZIP code and discriminated against people of color. Practices like redlining, that intentionally disadvantage based on protected features are referred to as disparate treatment. Disparate impact refers to seemingly neutral policies that are fair in form, but inadvertently discriminate against a protected group in operation [63]. For example, if a company would require their employees to exceed a certain body height to become a manager, this would screen out a disproportionate number of applicants for a job.

Competing Notions of Fairness

The specific notion of fairness is very dependant on the discipline. A decision may be considered fair, if the process by which the result was derived was independent of given variables, especially those considered sensitive. Such traits of individuals which should not correlate with the outcome are for example gender, ethnicity, sexual orientation and religion. However, what information is considered sensitive is very much dependant on the context. For example, gender and age information should be used in medical diagnosis but not in a recruitment process. Therefore certain definitions of fairness, and thus certain fairness metrics, are demonstrably inappropriate in particular circumstances. For example, if there is a legitimate reason for a difference in the rate of positive labels between members of different protected classes (e.g. incidence of breast cancer by gender) then statistical parity between model results for the protected classes would be an inappropriate measure. In philosophy, for example, fairness is often about making the morally correct decision. since morality is hard to compute, often a more quantitative perspective is taken. The focus is on constructing an optimal ML model under given fairness constraints

Fairness can be defined at the individual level, ensuring that similar individuals are treated similarly. The prediction probability $f(X, P = 0) \approx f(X, P = 1)$ has to be independent from the protected feature for individual observations. The same can be defined at the group level. In the latter case, this is done by grouping observations and ensuring that the groups are treated somewhat equitably. For this the expected prediction value $E[f(x, p = 0)] \approx E[f(x, p = 1)]$ has to be equal across protected and unprotected group.

2.1.2 Metrics

Algorithms are often undisclosed, proprietary, and constantly evolving, making them difficult to dissect and govern [59]. How to quantify fairness if the algorithm training data, objective function, and prediction methodology is given as a black box? For a specific predictor, learning to predict a target based on available features, fairness metrics help to quantify and uncover potential bias and discrimination. Since many different fairness metrics for ml and DL tasks exists and the most prominent ones will be mentioned here. In the following, X is the set of observations $\{x_1, x_2, \ldots, x_N\}$, Y contains the corresponding labels $\{y_1, y_2, \ldots, y_N\}$ and $P = \{p_1, p_2, \ldots, p_N\} \in \{0, 1\}^N$ signals the protected group membership for every observation. Figure 2.1 shows how the observations are divided into unprotected and protected groups based on the protected features.

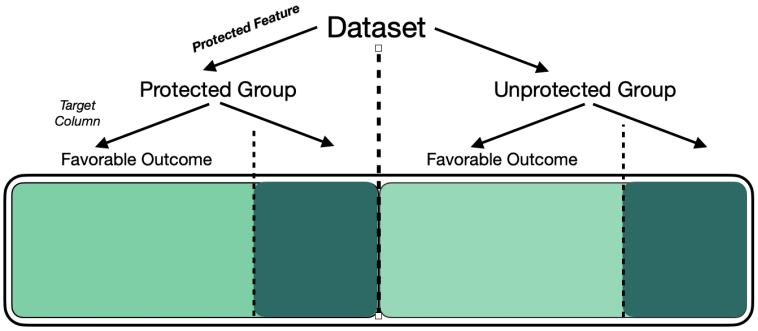


Figure 2.1: Data split into protected and unprotected groups

Question Fairness

Question fairness assesses whether the problem statement to be addressed, is inherently unfair. This can be the case, for example, when building a computer vision model to divide people into ethnic groups based on their appearance. This would be considered unfair in Western democracies. The Chinese government, on the other hand, does this systematically at the highest level and does not consider it an unfair practice. This metric is one of the most important, if not the most, and also one that is the most difficult to calculate algorithmically. Models built to answer unfair questions will never be fair.

Calibration

Disproportionate performance means that the model works substantially better for a subset of labels than for the whole set of observations. Many negative examples from the engineering domain exist. Since female crash test dummies were not required in the US, the crash safety systems performed significantly worse for smaller and lighter passengers, as they were optimized using taller and heavier crash test dummies. The goal of a well calibrated system is to ensure that the performance is equally distributed across groups [48]. A well calibrated system achieves a similar level of performance for every different type of stakeholder or observation.

Statistical Parity

Let X be a population and $S \subset X$ is a protected subset of that population. For example, X might be the set of all applicants that applied for an open position. S could be the subset of people amongst these applicants who colored their hair blue. A program that would recommend inviting applicants for an interview could discriminate against this protected subset by relatively inviting fewer people from the protected minority group, compared to the unprotected majority of people with non-blue hair. This discriminatory model behaviour could emerge for various reasons, for example from hidden bias in the training data. Checking for statistical parity (2.1) ensures that the same ratio of samples belonging to the minority and majority receive a favorable prediction ($\hat{Y} = 1$). The favorable prediction represents the set of desirable outcomes. With respect to the job application scenario, this could be the invitation to an interview or being admitted to the job. $Pr(\hat{Y} = 1|P = p)$ represents the probability of receiving a favorable prediction, if the sample belongs to the protected minority ($P = 1$) or the unprotected majority ($P = 0$).

$$Pr(\hat{Y} = 1|P = 1) = Pr(\hat{Y} = 1|P = 0) \tag{2.1}$$

In some cases, statistical parity is a central goal (and in some, it is legally mandated). Statistical parity, ensures that the overall proportion of members in a protected group receiving positive predictions is identical to the proportion of the whole population. Eq 2.2, also known as Demographic Parity (DP), requires the positive outcome to be equally distributed over the protected and unprotected group. Both segments should receive positive outcome at equal rates. This ensures that a model prediction does not dependent on the value of a sensitive attribute.

$$Pr(\hat{Y} = 1|P = 1) = Pr(\hat{Y} = 1) \tag{2.2}$$

$$SPD(\hat{Y}, Y, P) = |Pr(\hat{Y} = 1|P = 1) - Pr(\hat{Y} = 1|p = 0)| \tag{2.3}$$

Statistical Parity Difference (SPD) is calculated as the spread between the percentage of observations from the majority group receiving a favorable outcome compared to the protected group. If the SPD (eq. 2.3) is close to zero, the classifier is said to have "statistical parity", conforming to this notion of fairness.

Certain definitions of fairness, and thus certain fairness metrics, are demonstrably inappropriate in particular circumstances. For example, if there is a legitimate reason for a difference in the rate of positive labels between members of different protected classes (e.g. incidence of breast cancer by gender) then statistical parity between model results would be an inappropriate measure.

Equal Opportunity

Equal opportunity measures, whether the same ratio of people who qualify for an opportunity are equally likely to be selected, regardless of their group membership. The concept of equal opportunity originates from the context of hiring decisions. Hereby, meritocratic access, in which all qualified applications have a fair chance of attaining the respective position, should be assured. However, features that are considered to be protected in a society (e.g age, gender) should, have as little influence as possible on the outcome. It is calculated as the difference in true positive rate between the protected and unprotected group. The concept of equalized odds can be applied to target and protected attributes taking values in discrete and continuous spaces. For labels Y, predictions $\hat{Y}$ and protected attribute P, equalized odds are defined as shown in equation 2.4. A binary classifier satisfies equal opportunity with respect to P if it fulfills this equation.

$$Pr\{\hat{Y} = 1|P = 1, Y = y\} = Pr\{\hat{Y} = 1|P = 0, Y = y\}, y \in 0, 1 \tag{2.4}$$

$$EOD(\hat{Y}, P) = Pr\{\hat{Y} = 1|P = 1, Y = y\} - Pr\{\hat{Y} = 1|P = 0, Y = y\}, y \in 0, 1 \tag{2.5}$$

The equal opportunity difference is defined as the spread between TPR, FPR between the groups. For the outcome y = 1, the constraint requires that the classifier has equal TPR across the two demographics. For y = 0, the constraint equalizes FPR. Equalized odds thus enforce both equal bias and equal accuracy in all demographics, punishing models that perform well only on the majority. In order for the metric to make sense, it is very important to validate the assumptions beforehand. For equality of opportunity, you need to ensure that there are reasons why a positive predictive parity for two groups is desirable. In medicine, when recommending therapies, it may make sense to give male patients a different recommendation from female patients. In this case, this metric should not be applied.

Table 2.1: Sample confusion matrix from a binary classifier

Unprotected Group			**Protected Group**		
P=0	$\hat{Y} = 0$	$\hat{Y} = 1$	**P=1**	$\hat{Y} = 0$	$\hat{Y} = 1$
$Y = 0$	8000	3000	$Y = 0$	7000	9000
$Y = 1$	4000	6000	$Y = 1$	6000	9000

$$\begin{aligned}
TPR_{unproteced} &= Pr\{\hat{Y} = 1|P = 0, Y = 1\} = 6000/10000 = 60\% \\
TPR_{protected} &= Pr\{\hat{Y} = 1|P = 1, Y = 1\} = 9000/15000 = 60\% \\
FPR_{unproteced} &= Pr\{\hat{Y} = 1|P = 0, Y = 0\} = 3000/11000 \approx 27.27\% \\
FPR_{protected} &= Pr\{\hat{Y} = 1|P = 1, Y = 0\} = 9000/16000 \approx 56.25\% \\
EOD(\hat{Y}, P) &= \frac{9000}{16000} - \frac{3000}{11000} \approx 56.25\% - 27.27\% \approx 28.98\%
\end{aligned} \tag{2.6}$$

Computing the equal opportunity difference is illustrated in example 2.6 using a classifier with equal true positive rates across groups, but different false positive rates. This results in an equal opportunity difference of 28.98%.

Underfitting

Underfitting describes an AI-based model, when it is unable to capture the relationship between the input and output variables accurately, generating a high error rate on the training data and test set. This behaviour occurs when a model is too simple, which can be a result of a model needing more training time, more input features, more degrees of freedom or less regularization. Like overfitting, when a model is underfitted, it cannot establish the dominant trend within the data, resulting in training errors and poor overall performance. If a model cannot generalize well to new data, then it cannot be leveraged for classification or prediction tasks. The generalization of a model to unseen examples is ultimately what allows us to use machine learning in order to make predictions and classify data. High bias and low variance are good indicators of underfitting. Since this behavior can be seen during the training process, underfitted models are usually easier to identify than overfitted ones.

Overfitting

A model is declared to be overfitting when it achieves much higher performance on the data it has been trained on, compared to unseen or test data. This results in a significant negative spread between training and test performance. If the trained model predictions have high variance and low bias, it is declared overfitting [31]. Only if certain test performance is achieved at all (low bias), it must be checked whether the model is overfitting (excessively high variance) at all. Models that only achieve an insufficient test performance are not trustworthy anyways. This metric helps evaluate whether a model will be able to make good predictions for new data.

Average Odds

Average odds ensure that the classifier achieves equally good performance for the protected and unprotected group with respect to true positive (TPR) and false postive rate (FPR). The metric is computed as the mean absolute difference in false positive rates and true positive rates between protected and unprotected groups (2.7).

$$AOD(\hat{Y}, Y, P) = \frac{\Sigma_{y \in \{0,1\}} |Pr\{\hat{Y} = 1 | P = 1, Y = y\} - Pr\{\hat{Y} = 1 | P = 0, Y = y\}|}{2} \quad (2.7)$$

The concept originated in the field of sports betting with binary options (for example, whether team A or B wins), as it is also about a trade off between risk and return, which is reflected in the TPR and FPR. The metric takes on values ranging from 0 to 1. The closer the average odds difference is to zero, the more evenly distributed the model's performance is between protected and unprotected groups.

Disparate Impact

Disparate treatment refers to the practice of intentionally factoring protected attributes into an algorithmic decision-making process, such that different subgroups are treated differently. The concept originated from the context of employment decisions and conviction [63]. An example of disparate treatment would be an employer using information about an applicant's vaccination status to exclude non-vaccinated applicants. Federal laws prohibit such discrimination based on race, color, sex, sexual orientation, gender identity, national origin, religion, age, disability and genetic information. It is unfair, if members of a protected group systematically and unjustifiably receive worse predictions. This phenomenon is known as disparate impact. Disparate Impact (DI) is computed as the ratio of samples from the protected group receiving a favorable prediction divided by the ratio of samples from the unprotected group receiving a favorable prediction (eq. 2.8).

$$DI(\hat{Y}, Y, P) = \frac{Pr\{\hat{Y} = 1|P = 1\}}{Pr\{\hat{Y} = 1|P = 0\}} \tag{2.8}$$

DI occurs when an organization's actions, policies, or some other aspect of their processes inadvertently result in unintentional discrimination against people who are in a protected class. Even though the policy, action, or item in question would otherwise appear to be neutral. What matters is the outcome, not the intent. In order to avoid a disparate treatment of minority and majority, the decision must be statistically independent from all protected attributes. The protected features vary depending on the context. Due to a potentially existing correlation between features, it is not enough to just exclude sensitive attributes from decision making. For example, removing sensitive demographic attributes from a training data set that still includes postal code as a feature may address disparate treatment of subgroups, but there still might be a disparate impact upon these groups because postal code might serve as a proxy for other demographic information.

Class Balance

In many application scenarios (such as fraud detection or disease screening) machine learning is used in order to detect rare exceptional cases. However, classes occur with different frequencies. The exceptional case naturally occurs less frequently than the normal case. This statement can be generalized to multi class classification. Therefore, the class imbalance is a common problem. Imagine that a rare heart disease (prevalence $<$ 1%) is supposed to be predicted based on patient data. A model that always delivers a negative test result would achieve an unjustified accuracy of over 99% due to a strong class imbalance. This is an issue since many predictive models assume that the target classes share similar prior probabilities. Decision trees for example are vulnerable to imbalanced data, since they tend to achieve better performance for the majority class [19]. Therefore, a disproportionate ratio of observations in each class can lead to unfairness. Stratified sampling can be used during data acquisition in order to ensure an equal frequency with respect to features or labels. Afterwards, oversampling the minority class or undersampling the majority class, can compensate for imbalanced data [35].

2.1.3 Limitations

Fairness in machine learning is a growing field that has not yet received enough attention. Model developers, as well as users, need to be aware, that AI systems are susceptible to bias and can therefore exhibit discriminatory behavior. It is especially difficult because the concept of fairness is based on cultural values and emerges as the result of social discourse. Depending on the actual circumstances and scenario, a result may be considered discriminatory at one time and acceptable at another. This is also related to the existence of the many different definitions of fairness (e.g. individual vs group fairness [14]). Some metrics, like question fairness for example, are infeasible to be computed. Therefore, model developers and operators must ensure that their model is used in non-discriminatory and fair manner. In order to prevent legal disputes, it is very important as a company use AI within the prescribed legal framework about fairness.

2.2 Explainability Pillar

In the context of ML, explainability means that an interested stakeholder can comprehend the main drivers of a model-driven decision [15]. At basic definition of explainability, is to understand the functional relationship between input and output. For simple ML models, like linear regression or logistic regression, the functional mathematical dependence is simple enough such that it can be directly understood by humans. Also for example the behavior of a small decision tree is still comprehensible. For simple models, we can understand each detail and follow the decision process of the model and validate it. However, this is not the case for more complex models like neural networks or random forests, where the relationship between input and output can be way too complicated to understand. In those cases, one is not able to understand the algorithm in detail, but one can observe behavior in the output with changes in the input and get an estimation of the relationship between them [33].

Fully understanding the nature and inner workings of an algorithm has become a desired goal for ML models. Such models are often referred to as interpretable models, which are especially important in domains such as criminal justice, medicine or business. In these domains, AI often acts as support to human decision-making. Such models could for example be used by juries for the recommendation of sentences, or to support bankers in the decision of loan lending, or doctors with diagnosing diseases. Since those decisions can have a major impact on human lives, transparency is needed. When it comes to trusting, understanding and criticizing such models reasoning of an individual decision is required [21].

The area of explainable artificial intelligence (XAI) is focused on gaining a deeper understanding of the inner workings of AI-based systems [12]. The goal is to make the behaviour of AI systems more understandable to humans by providing good explanations about what it has done, what it is doing now, and what will happen next. The explanations can either reveal full or partial information about the underlying process. Explanations can also be global and local, where global ones focus on the general behaviour of the model and local explanations focus on understanding a single input output pair [34]. Among ML models there is a distinction between models that are interpretable by design and those that can be explained with the help of external post-hoc interpretability techniques like sensitivity analysis, tree interpreters or neural network interpreters. Among post-hoc explainability techniques, it can be distinguished between text explanations, visualizations, local explanations, explanations by example, explanations by simplification and feature relevance [12]. The focus of this project is to analyse the level of global model explainability and not on post-hoc explanations, which is why no further details are provided about those techniques.

2.2.1 Motivation

In order to solve complex computational tasks (e.g NLP) more complex models such as deep neural networks are used and can achieve an astonishing performance. Such models often have millions of parameters and are called black-box models, because the mechanism of how the model works can not be directly understood. Although such models can solve complex tasks, there is a well-known trade-off between the model's performance and explainability [24]. The relative explainability of different types of models is illustrated in Figure 2.2.

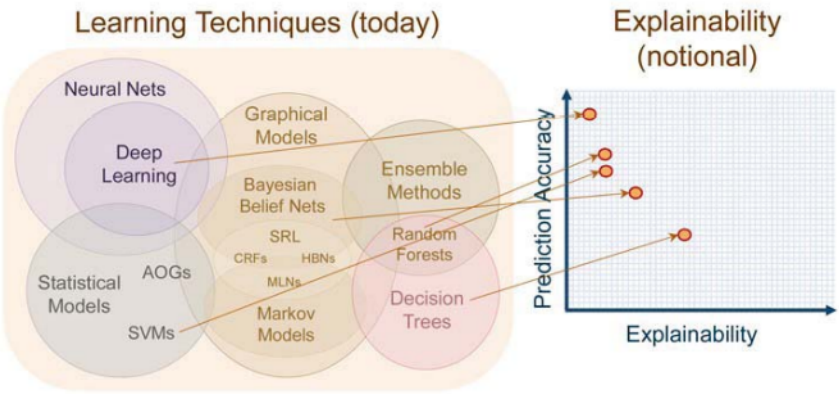


Figure 2.2: Relative explainability of different ML algorithms [23]

Therefore, solely focusing on increasing the performance, can be dangerous as it can create decisions that are not explainable nor justifiable. As AI models and services are becoming more prevalent, there will be compliance and regulatory needs, asking for information about the behavior of those models and the ability to explain a particular decision [38]. See in Figure 2.3, who might be interested in such explanations.

When it comes to regulation the European General Data Protection Regulation (GDPR) states, "the existence of automated decision-making should carry meaningful information about the logic involved, as well as the significance and the envisaged consequences of such processing for the data subject [26]". This passage underlines the importance of XAI.

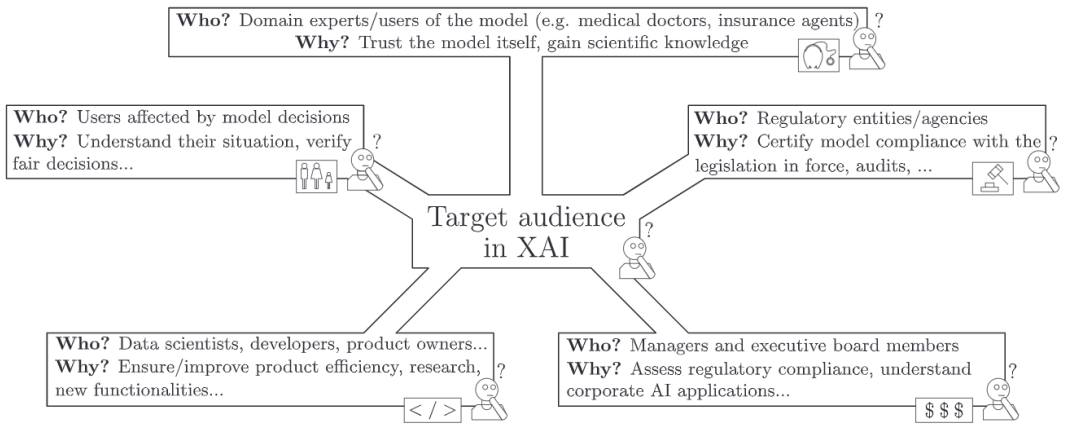


Figure 2.3: Different purposes of XAI [12]

XAI's key message is that trust needs explainability. For example, financial institutions need to be able to comprehend their automated money lending decisions and possibly explain them to the lender, the loaner and maybe even the regulators. A customer might want to know why his loan was rejected or what would have to improve for the loan to be granted and he could even have the legal right to get that information [18].

2.2.2 Metrics

This work puts a focus on global explainability, which is about understanding the general behaviour of the model. As for now, there is no predefined set of metrics that can capture the global level of model interpretability. This chapter collects knowledge from the literature about model explainability as a set of metrics. Model transparency is clearly defined in the literature and is strongly linked to global model explainability. Since transparent models are also explainable by design, metrics from the sphere of ML transparency were considered as well.

Model transparency comprises three main aspects: simulatability, decomposability and algorithmic transparency [12]. *Simulatability* requires that a human being can simulate the model in his head and think and reason about the model as a whole. For this characteristic model complexity plays a dominant role. *Decomposability* means that every part of the model (model inputs, parameters and calculation) must be comprehensible by humans. This for example requires that the features are readable and not highly engineered so that it is clear what each feature represents [85]. *Algorithmic transparency* means that a model can be fully explored with the help of mathematical analysis. It is about mathematically understanding the process mapping inputs to outputs [44].

Below follows a list of four metrics for machine learning classification models which is an attempt to capture the most important factors when it comes to the general level of explainability. These metrics are inspired by scientific literature and together constitute a first sample taxonomy for global model explainability.

1. Algorithm Class

The inherent complexity of the specific learning algorithm has a strong influence on the final model explainability. This metric evaluates to what degree the model is explainable based on the used learning algorithm. The used learning technique is the main driver for model complexity and therefore has a big impact on transparency and explainability. The learning technique affects the whole inner working of the model and already confines the set of possible post-hoc explanations. It is therefore the most important metric to consider. The following analysis of different ML models contains information and reasons for how well the model decisions can be understood and explained.

Logistic Regression (LR) is one of the oldest and most used classification algorithms and is among the simplest unsupervised learning techniques. It assumes a linear relationship between the dependent and the independent variables which makes the fitted coefficients easily interpretable. LR meets all three characteristics of explainable models. The mathematical formulation is transparent and the results can also be simulated and decomposed [47].

However, the closed-form solution of LR can be difficult to understand. It can also be dangerous to interpret the coefficients or compare them to other LR models with different features since the coefficients may also include unobserved effects which vary between different models and the assumptions of LR models, in reality, do often not hold. Even if they hold it can still be difficult to interpret the coefficients if the inputs are highly engineered.

But Simulatability is always guaranteed for LR since the coefficients are known and For any given input a human can calculate the output from the coefficients and understand the working of the model. Since LR requires expert knowledge to correctly interpret the coefficients, the global level of explainability of LR models is on a medium to a good level.

Decision Trees (DT) recursively partition data by the highest information gain. At every step, the remaining features are evaluated and the feature that best splits the data is selected as a node. Given a fitted DT, the output of the model can be easily simulated by humans for any input by just following the branches on the tree. The information gain can be interpreted as features importance which makes the features at the top the most important ones. This means the resulting tree itself is already very comprehensible and explainable by design if it is not too deeply branched and the features are understandable [55] .

A DT has every characteristic of transparent models and is even used for rule extraction. For example, DT induction is used to help interpret black-box models where the behavior of the black-box model is learned and represented with a DT [71]. A DT is understandable by everyone and is without a doubt the most transparent model. Therefore this algorithm is very well explainable.

Random Forest (RF) are among the most accurate classification models, constructing multiple independent DTs at training time and aggregating prediction results using a voting schema. The combination of multiple independent DTs makes the interpretation more complex [12]. Although the composite model is not as explainable as a whole anymore, the relevance of each feature can still be computed. Since an additional aggregation layer is added to the explainable DT model, random forests are only moderately explainable [16].

Bayesian Models fall under the hood of transparent models. They describe conditional dependencies of the features and the output. This provides a high level of explainability, as the probabilities represent a clear relationship between the input and output. If the Bayesian belief network is not too complex it tends to be simulatable, decomposable and algorithmically transparent. However, if the network grows bigger or someone is not familiar with probabilities, the first two properties might get lost [12]. There are also other models based on the Bayes theorem like Markov chains [57], which are less explainable since they contain a stochastic nature. But all Bayesian models tend to be explainable and intuitive because their solutions can be well represented graphically.

K-Nearest Neighbors (KNN) is another method the falls within transparent models, meaning they are simulatable, decomposable and algorithmically transparent. The decision of the model is based on the most similar cases, which is comparable to how humans make decisions based on experience. To measure the similarity a distance function is used, but if this function is too complex or a high value for K is chosen the method is not simulatable and transparent anymore [43]. The output of the model can be simulated for any input, but understanding the general behavior and working of the model is difficult. For a local case, the nearest neighbors can be identified, that influenced the classification, but it is not possible to tell which features are most important on a global scale. This algorithm is therefore only moderately explainable.

Support Vector Machines (SVMs) are a type of ML algorithm that constructs a hyperplane, that can be used as a decision boundary. The plane has the largest distance to the nearest data point of any class [36]. SVMs are complex and the equations of the kernels are difficult to understand. The support vectors by themselves cannot be interpreted and one does not know which features were important in calculating them. There exist many post-hoc explanation techniques for SVM but they are not well explainable by design [12].

Artificial Neural Networks (ANNs) try to mimic the biological structure of a human brain, with many connected neurons that can get activated. There exist different types of neural networks like multi-layer neural networks, convolutional neural networks or recurrent neural networks, and they all can infer complex relationships between the features. A neural network can have millions of parameters constructing a large network with many layers and nodes [83]. It is the least explainable machine learning method since every node represents a nonlinear relationship of all previous nodes. For a human being, it is not possible to reason about the working of the algorithm and understand its behavior. This is why they have always been considered black-box models [64]. Explainability techniques for those models do exist, which use for example model visualization, model simplification or feature relevance estimates, but they often only work on neural networks with one single hidden layer. Even with those techniques, neural networks provide if at all a very low level of explainability and they do not count as transparent models.

Q-learning is an unsupervised learning technique and learns from positive and negative rewards. From a given state it estimates the future rewards for every possible action and chooses the action with the highest expected reward. The behavior of this model is very hard to explain and understand for humans since the agent is always looking multiple steps ahead and considers every possible action. With a given q-table humans can simulate the outputs for any given inputs, but it is not possible to understand the relationship between input and output. Also algorithmic transparency is not given for unsupervised learning techniques and therefore this class of algorithms provides a very low level of explainability.

Summary In conclusion, the most important ML algorithms have been analyzed on their level of explainability. Counting as transparent and explainable algorithms are decision trees, Bayesian models, logistic regression and k-nearest neighbors where decision trees are the most explainable and k-nearest neighbors the least explainable. Support vector machines, q-learning and neural networks are considered as black-box models and their calculation is not transparent and their explainability is questionable. With the knowledge, it is now possible the classify ML models according to their level of explainability on behave of their used algorithm.

2. Model Size

This metric is based on the number of parameters a model is using. The number of parameters correlates with the input dimensionality of logistic regression or the number of edge weights and biases in a neural network. This metrics is used in a majority of papers when it comes to evaluating models on the level of comprehensibility. The underlying assumption is that the comprehensibility inversely correlates with the models Degrees of Freedom (DoF) determined by the number of parameters. However, only looking at the model size, in order to determine the level of explainability, would not be enough [30]. It does not capture any semantic, which means that as a stand-alone metric for explainability, it would not be very useful, since a big decision tree is still more interpretable than a small neural network. But when we take a ML model and only vary the number of input features the smaller model will be more comprehensible. Also for transparent models, to maintain the characteristics like decomposability and simulatability the model must not be too large. Otherwise, it will not be possible for humans to simulate and think about the model as a whole [12]. Therefore as an addition to the algorithmic class metric, model size can be taken into account when measuring the level of explainability.

3. Correlated Features

This metric counts the number of highly correlated features. Ideally, the input features are not correlated with each other and highly correlated features have been dropped. Having a high correlation among features leads to biases in most explanation techniques, especially when trying to represent feature contribution. For example, partial dependence plots are often used for showing local feature importance by showing the marginal effect of one feature on the predicted outcome of the model. But if two features are correlated

and one of them is permuted while all others are kept the same, the model still has the other correlated feature present. Therefore the outcome would not change much and the marginal effect would be biased. Important features would not be seen as important because of the correlation. Many ML models also assume that the variables are uncorrelated. If this assumption does not hold, once interpretable models can not be interpreted anymore. Generally, creating explanations with highly correlated features is difficult, as it can not be distinguished between the effects of correlated features and that leads to biased explanations.

4. Feature Relevance

Looking at the relevance of each feature is currently the most popular explanation technique. Feature relevance measures the individual contribution of a certain feature to the final outcome, regardless of the direction (negative or positive) or shape (linear or nonlinear) of the effect [68].

Feature relevance can be divided into local importance, which measures the relevance of the features for a specific observation or global importance which measures the relevance of the features of the entire model. Local feature relevance can be measured with various local explainability techniques. Many local techniques also work on black-box models, by simply observing how much an output changes with small permutations in the given input and find out which features have a strong effect on the resulting output.

But since we are interested in global explainability we are only interested in the global feature relevance, which cannot be calculated on black-box models as the features input space is either infinite or very close to infinite. Global feature importance is dependent on the model type and is given from the model and not calculated with external techniques, which could result in different feature relevance scores. An example for global feature relevance could be the coefficients in a logistic regression or the importance of the nodes in a decision tree or the assigned probabilities in a Bayesian belief network. Global features relevance can therefore not be calculated for all models, for example, SVM provide no information about which feature was important for the resulting decision boundary and neither do k-nearest neighbors or neural networks.

If the global relevance of the features can be extracted one can analyze its distribution. For meaningful explanations, there should not be any irrelevant features as they would only make an explanation more complex without being relevant. This metric is therefore analyzing the distribution of the feature relevance scores.

2.2.3 Limitations

The explainability of a ML model depends on the specific stakeholder, who wants to make use of it. For example, the coefficients of a Bayesian model can help an expert in understanding the inner workings of the model, but to someone who is not familiar with Bayesian models, it would mean nothing. Explaining models to stakeholders with little mathematical background is a general difficulty.

XAI has to be designed to explain itself to its users in a comprehensible way, for example with graphical illustrations or textual explanations. There should be a layer between the model and the user in form of some kind of explanation interface. There exist many techniques to create explanations and several metrics exist to elaborate the goodness of explanations or whether users are satisfied by the explanations [39]. An explanation is never just there but needs to be created and extracted from the model. This translation could be done by dozens of explainability techniques and cannot be captured from the model alone.

This means that even if explainable ML algorithms are used but no explanation layer is created, the model is not explainable to end-users. It can also be the other way around, where non-explainable models like neural networks can be designed in such a way that the overall behaviour of the model can be explained with neural network interpreters. Depending on the model, this additional explanation layer can vary, but in order to get an explanation, it must be done.

An AI system must be explainable by design. For example, it is crucial to use meaningful, understandable and simple features. Otherwise no matter how transparent and explainable the model might be if it is not clear what a certain feature represents the model can never be understood.

Algorithms are not explainable by themselves, but additional work is required to create meaningful explanations that are tailored to the knowledge of the end-user. Choosing explainable algorithms is not enough, but the model has to be designed to be explainable.

2.3 Robustness Pillar

In the context of ML, robustness refers to an algorithm's ability to deal with attacks based on adversarial examples. ML models are not perfectly robust against malicious intents. Both, simple statistical models and complex deep neural networks are vulnerable to adversarial attacks. This is focused on shifts in the output of the algorithm, that are caused by slight perturbations of the input data. If small changes in the inputs can cause big deviations in the output, it means that adversarial perturbations can be used to generate undesired outcomes and the classification process is not robust [5].

On the other hand, the robustness of an ML algorithm can relate to its ability to generalize. It then characterizes how effective the algorithm is, when being tested on a dataset, having a different statistical distribution than the training dataset. If the algorithm is robust, the results on slightly different data should be comparable and should not create a big unexpected shift in outputs. Robust machine learning algorithms have smooth decision functions that output bounded and stable results [3].

2.3.1 Motivation

Machine learning models are being designed to interact with the real world. In the real world, one cannot control every single input that is given to the model. Models are trained with selected and labeled data but they have to perform with unseen real-life inputs. The training data will always be just a subset of all theoretically possible inputs values. As an example, a self-driving car may identify the moon as a yellow traffic light [78]. One can assume the training data of the car algorithm does not contain the images of the moon from that specific angle. Unfortunately, the algorithm failed to generalize the known data to the physical world. This is an alarming scenario in terms of robustness. One must be sure that the algorithm runs as expected given any input.

Hiding model details, like parameters, is not enough to protect the model against attackers. Even in the black-box setting, models can be deceived with optimized small perturbations on the inputs. The authors [6] demonstrated that the automatic recognition of stop signs can be decreased by 80% by purposefully attaching small stickers to them. In the context of self-driving vehicles, such performance degradation due to small perturbations can be extremely dangerous. The car could simply miss a stop sign and thus get into a dangerous situation.

The output of the machine learning algorithms should be stable and bounded. We cannot trust algorithms that return unexpected outputs. Physical-world can create unpredictable data, we need models durable enough against uncertain data. Hence when we are talking about trusting machine learning models we cannot disregard robustness.

2.3.2 Metrics

Loss Sensitivity: We previously mentioned that robustness is related to the shifts that can be generated in the output when we apply small perturbations to the inputs. If slight

changes in the inputs may cause big deviations in the output, we would say that the model is not robust [5]. Local loss sensitivity measures the largest variation of the output of the model under a small change in its input. The smaller the variation in the output, the smoother the function. Overall it quantifies the smoothness of a model.

Figure 2.4 visualizes 4 different models' decision surfaces that are all trained on the CIFAR dataset. The 2 models on the right are considered to be more robust as their function surface is smoother [3].

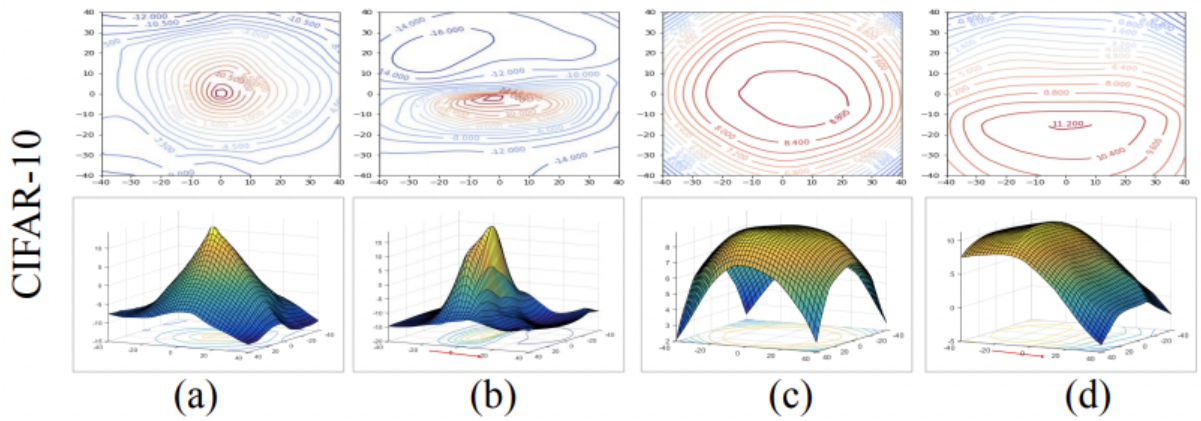


Figure 2.4: Different function surfaces illustrating loss sensitivity [3]

$$g = \left\| \frac{\partial \mathcal{L}}{\partial x} \right\|_1 \tag{2.9}$$

Loss sensitivity is defined as shown in the equation 2.9 where $\mathcal{L}$ is the loss function. Here l_1 norm is used in the definition but changing the norm would not change the results significantly. The loss sensitivity can be calculated using the gradients of the prediction for given test data [7].

Confidence Score: for a robust classifier, the decision function should have a smooth neighborhood. This implies that a robust model should be stable on its predictions such that the predictions don't change immediately with small changes. Confidence score measures the stability of predictions [3]. The more stable the predictions are, the more robust the model is.

The confidence score is defined as the prediction probability rate. As an example to give more intuition, if a model says that it is 52% sure about its prediction, it can be deceived easily. On the other hand, a model with a 98% prediction probability would be difficult to fool.

CLEVER Score: is a robustness evaluation metric developed for neural networks and it is an abbreviation for Cross Lipschitz Extreme Value for Network Robustness [8]. It uses the Lipschitz constant in order to estimate an upper bound on changes in the output with respect to input perturbations. Figure 2.5 illustrates the theorem behind the CLEVER score. The focus lies on the classification functions output $g(x_0 + \delta)$ where x_0 is a known data point and δ represents a small perturbation. The theorem states that this output

is upper-bounded by $g(x_0) + L_q||\delta||_p$ and lower-bounded by $g(x_0) - L_q||\delta||_p$ [8]. Here L_q denotes the local Lipschitz constant and $||\delta||_p$ is the l_p norm of distortion δ with $p \geq 1$.

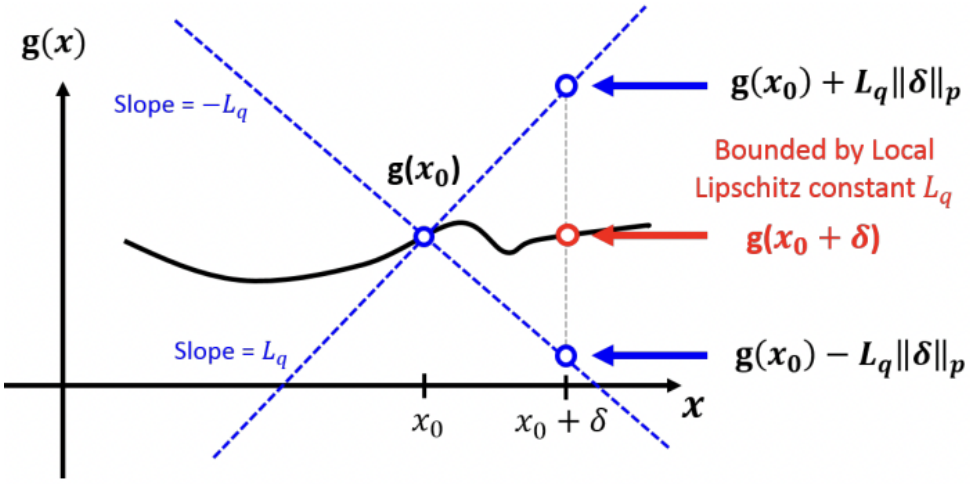


Figure 2.5: The main intuition behind using Lipschitz constant on robustness estimation [8]

For a given input, the CLEVER score estimates the minimal perturbation that is needed to change the classification outcome, using the local Lipschitz constant [8]. This allows assessing a model's local robustness.

Clique Method: was developed for tree-based models. Decision trees are built using non-continuous step functions. Therefore, gradient-based metrics don't work for tree models. The robustness verification problem in trees can be reformulated as a max clique finding problem [4]. Instead of focusing on the calculation of minimal adversarial perturbations, Chen et al. focused on the robustness verification problem shown in the equation 2.10 [4].

$$Does\ there\ exist\ an\ x' \in Ball(x, \epsilon)\ such\ that\ f(x') \neq y? \tag{2.10}$$

$Ball(x, \epsilon)$ is the collection of the points that are at most ϵ away from a given example x and ϵ sufficiently small. $f(x)$ is the classifier's prediction on data x and y is the correct class label for x. If we can say no to this question for selected ϵ and x then we would say that our model is robust around x.

The core idea of the tree robustness verification is finding a ball for each leaf such that any x in this ball will fall into that leaf node. Since at each level, the thresholds for decision making are already known, one can find the mentioned ball via applying depth-first traversal. Starting from this point for single decision trees the idea can be extended to be applied to tree ensembles. It is proven that for tree ensemble models, the same verification problem can be reformulated to be equal to the maximum clique finding problem (where a clique is defined as a fully connected subgraph) [4]. After this reformulation, we can use existing efficient maximum clique searching algorithms.

Empirical Robustness Assesses the robustness of a given classifier concerning a specific attack and test data set. It empirically tests the algorithm against a chosen attack. It

corresponds to the average minimal perturbation that needs to be introduced for that type of attack to be successful [7].

For the calculation of Empirical Robustness, an attack type needs to be chosen. Then having normal inputs, we need to create multiple adversarial examples via perturbing the normal inputs according to the parameters of that attack. Then we will compare the outputs of the model to the normal test data against the outputs of the model to the adversarially crafted inputs. If the model performs significantly worse on the crafted inputs, it means the attack was successful on the algorithm. Hence the algorithm is not robust enough against that attack.

$$ER(C, \rho, X) = \frac{1}{|I|} \sum_{i \in I} \frac{||\rho(x_i) - x_i||}{||x_i||} \tag{2.11}$$

Empirical robustness can mathematically be defined as in equation 2.11 where C is a trained classifier, ρ is an untargeted attack [7]. $X = (x_1, x_2, ..., x_n)$ are given test data. First adversarial inputs $\rho(x_i)$ are being crafted and the classifier is tested against them. In the equation above only the adversarial inputs which successfully fooled the model are being considered. So only the indices $I \in 1, 2, ..., n$ where $C(x_i) \neq C(\rho(x_i))$ must be taken [7].

Selecting attacks to test the model is a challenging task. In this project, Fast Gradient Attack, Carlini Wagner Attack, and Deepfool Attack are chosen. Our decision is mainly based on the efficiency, success ratio and calculation speed of the attack. Fast Gradient Attack was chosen because it is well-known to be effective and it doesn't require much time for calculations [5]. Carlini Wagner Attack is a state-of-the-art attack that is very successful [56]. Deepfool Attack is also fast and accurate [53].

Empirical Robustness - Fast Gradient Attack: In their paper Goodfellow et al. came up with a new method to create adversarial examples using gradients which later will be referred to as the "fast gradient method" [5]. It is a white-box attack which means for the calculation, one must know the structure and the parameters of the algorithm. More specifically for this attack, we need to know the loss function of the machine learning model. Adversarial examples are calculated as in the equation 2.12 [5].

$$X_{adversarial} = X + \epsilon \times sign(\nabla_X \mathcal{L}(X, Y)) \tag{2.12}$$

Here X is known test data, ϵ is a small step size and $\nabla_X \mathcal{L}(X, Y)$ is the gradient of the loss function of the classifier with respect to X. So starting from a test input X we try to go one ϵ-sized step towards the gradient direction [5].

The equation is a gradient ascent on the loss function. The intuition behind applying gradient ascent is to increase loss and hence the error of the algorithm. Here we cannot update the model parameters thus we update the data to imitate a similar behavior. The attack is very straightforward, practical, easy to implement, and effective.

Goodfellow tested the attack that they propose on a logistic regression model to present the accuracy of the metric. The logistic regression model was trained on the MNIST data

set and initially had an error rate of 1.6% on normal test data. On the calculated fast gradient attack examples the model had an error ratio of 99% [5].

This attack can be calculated for neural network models with accessible and differentiable loss functions. It also can be calculated for logistic regression and support vector machines because their classification functions are sufficiently simple and differentiable [5].

Empirical Robustness - Carlini Wagner Attack: This attack was named after Nicholas Carlini and David Wagner who proposed the attack. It is a targeted attack meaning the attackers choose a target class and try to perturb any input to be classified as that target class [7].

Attack design immediately follows from the robustness definition. They come up with the optimization problem expressed in the equation 2.13.

$$\begin{aligned} &\text{minimize } D(x, x+\delta) \\ &\text{such that } C(x+\delta) = t \\ &\qquad x+\delta \in [0,1]^n \end{aligned} \tag{2.13}$$

Where D is a distance metric such as l_2, l_0 or l_∞, x is an example input, C is a trained classifier and t is the target class. Here the problem describes that we want to find a minimum δ change on x which will make the classifier misclassify input as target t [56]. However, solving this optimization problem for any classifier is not easy as the functions are not linear in most cases. Hence they propose to change the optimization problem to be as in equation 2.14.

$$\begin{aligned} &\text{minimize } D(x, x+\delta) \\ &\text{such that } f(x+\delta) \leq 0 \\ &\qquad x+\delta \in [0,1]^n \end{aligned} \tag{2.14}$$

Here they define an 'objective function' f such that $f(x+\delta) \leq 0 \iff C(x+\delta) = t$. Definition of f is relatively flexible and there are several possible options [56]. However of course this approach assumes a white-box setting where the attacker knows about the model structure. After this construction one can solve the optimization problem using efficient algorithms to create adversarial examples.

Carlini and Wagner tested their attacks on a Neural Network model trained on the MNIST dataset with 99.5% accuracy. They achieved a 100% attack success ratio demonstrating that the attack is quite powerful. However as we can see in the comparison from the paper, it is often slower than the Fast Gradient Attack[56].

One can implement an untargeted version of this attack via simply modifying the optimization problem to aim the change of the original classification, rather than a defined target [7].

Empirical Robustness - Deepfool Attack: Deepfool attack is an efficient attack designed for deep neural networks. It is a white box attack, one needs to know the classification function of the model[7].

The theory behind the attack starts with a minimization problem. The perturbation Δ that should be applied on a given input x is defined as in equation 2.15 [53].

$$\Delta(x, k) = \min_{r} ||r||_2 \text{ such that } k(x + r) \neq k(x) \tag{2.15}$$

Here $k(x)$ is the predicted output of the classifier for given x. This optimization problem tries to find a minimum change in l_2 norm which will change the prediction of the classifier. This minimization problem and the one discussed for Carlini Wagner attack equation 2.13 are approximately the same. In Deepfool attack we only consider l_2 norm whereas equation 2.13 is a general form. Again as a different approach to the same problem, here we will assume that $k(x) = sign(f(x))$. From now on only the case of binary classification will be explained, however the method described can be generalized to all classifiers. The assumption on $k(x)$ will simplify to be a calculation of an orthogonal projection as shown in figure 2.6[53].

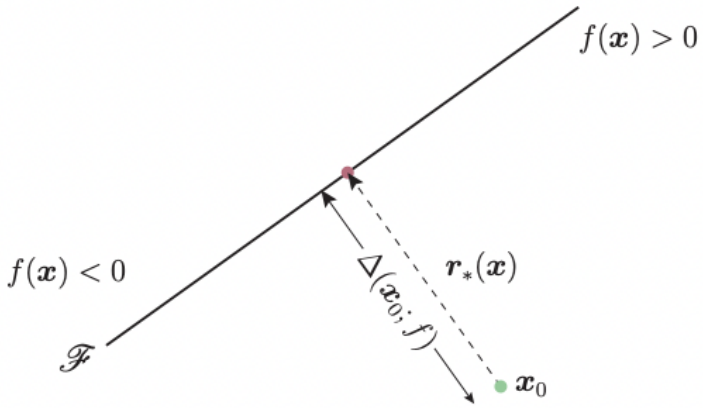


Figure 2.6: Deepfool attack approximation [53]

Figure 2.6 shows our simplified classification function $f(x)$. x_0 is an example point. As we are only considering binary classification, if $f(x) < 0$ then the classifier predicts one class and if $f(x) > 0$ it predicts the other class. We would like to change the prediction for a given input x_0. To do that all we need to do is to project our point x_0 onto our line $f(x)$. This would calculate the minimum distance to fool the classifier[53]. Calculation of orthogonal projection is easy to implement and efficient.

As mentioned before, this method can be extended to multiple dimensions and multiple classes. To decrease the effects of the global assumption on $k(x)$, they instead assume it for the classification function locally around the given point x_0 [53].

To summarize, given a classification function and an input x_0, we linearly approximate the function around x_0. Then we project x_0 onto the approximation to find minimum perturbation necessary to misclassify it. In the paper they iteratively apply the method to increase the accuracy [53].

2.3.3 Limitations

An adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction. Hence robustness is defined on models where the "False Prediction" is clearly defined and where true and false predictions are sufficiently different [7]. While the robustness of classification models is straightforward, it is quite difficult to talk about the robustness of regression models. For example, in the case of anomaly detection, without having access to the class labels, is a clustering problem. In an unsupervised outlier detection problem, it is difficult to have a clear definition of true-false predictions. However, by reformulating the unsupervised anomaly detection scenario into a supervised classification problem, by adding labels to samples, then robustness can be considered.

Robustness makes more sense if the input size of the model is sufficiently large. When the input is perturbed, the difference should be small, but still sufficiently large enough to cause a misclassification. This is easier for high-dimensional input [7]. Therefore, CNNs are often used as a reference in literature.

Some metric specific limitations should also be mentioned here, for example, that most metrics can only be applied to certain ML algorithms. Loss sensitivity and the CLEVER score were developed for ANN and cannot be used to test other algorithms [8]. The clique method is a tree-specific metric [4]. Since, for the calculation of the confidence score only prediction probabilities are needed, it can be calculated for all algorithms which return prediction probabilities [3]. Finally, the attacks that are chosen for empirical robustness are compatible with SVMs, logistic regression, and ANNs.

Although it is intuitive to use the confidence score as a robustness metric, it might not capture the full complexity of a robustness evaluation. It can be too simple to accurately measure and represent the robustness of complex machine learning algorithms [3]. Therefore, one might want to use additional metrics to support the analysis.

To execute, the Carlini Wagner attack a simplification function is applied to the objective function to allow solving the optimization problem. This simplification function must be chosen by the attacker. The choice of this function influences the performance. Even though in the paper multiple ways of choosing this function are discussed, they all require knowing the model's objective function [56]. If the approximation is not selected wisely, choosing the correct adversarial examples might not be successful.

The Deepfool attack is based on approximating the objective function. This time the approximation is constant and the user cannot intervene in the performance of the attack, but still one must consider that it is an approximation and may not perform well if the original objective function is very complex [53].

2.4 Methodology Pillar

To decide, if a ML model is supposed to be trusted in certain scenarios, one must understand how the model was built and evaluated. For this purpose, the model creation process should be documented as transparent and faceted as possible. Within the methodology pillar, it is checked and validated if important decisions taken throughout the ML model lifecycle are properly documented and communicated. Methodology score describes how a model is trained and validated. It confirms if the decisions taken are inline with the known best practices. Having this information at hand is supposed to increase users' trust in the system.

2.4.1 Motivation

Methodology is about documenting and validating the process used for creating and maintaining a model. This includes checking whether best practices were followed or not. Following best practices usually increase performance and improves reliability. As a stakeholder, having the most important information documented about a model, increases trust.

The decisions taken throughout the process of the training and the evaluation, must be considered as part of the trust score. Appropriate data pre-processing steps decreases outlier effects and increases prediction stability. Knowing that, we cannot ignore data preprocessing on the calculation of trust score. Also the communication of the details of the algorithm to the end users must be considered as well. It is impossible to trust a model when we are not provided with the structure of the model.

Knowing about the evaluation of the algorithm is crucial for the trust as well. For example considering train-test split, one can observe that standard performance metrics lose their meanings in the case of a unusual train-test split. If the model is trained on 95% of the data and tested on only 5%, the test data can be too small to represent the performance of the data. Checking only the performance metrics like accuracy or F1 score, we would miss the poor methodology and mistakenly trust the model.

2.4.2 Metrics

Normalization: Applying pre-processing on the data before training a machine learning model is crucial. One of the most important pre-processing steps is normalizing the data. In literature, multiple normalization methods, such as Z-score normalization, min-max normalization, and median normalization are described [77].

Z-score normalization also can be referred to as "standardization", which transforms each feature to have zero mean and have unit variance. It is known to reduce the effect of outliers on the algorithm [77]. Min-max normalization re-scales all values to fit into a given range, usually 0 to 1 or -1 to 1. The important property of min-max normalization is that it retains relative distance relations existing in the data [77]. Finally, the median

normalization is the simplest technique, subtracting the feature mean from each feature value.

It is known that normalization improves the performance of machine learning models and it is considered a best practice. We also claim that normalization is important for trust and we choose it as a metric. Because normalization helps mitigate the effects of outliers and makes sure that features take on values in the same range. Therefore, it is important to consider whether a model was trained on (non) normalized data when talking about trust.

Missing Data: This metric checks and evaluates how missing values are handled. If the model is trained on a dataset containing missing values, then the model will not be reliable. Another bad practice would be filling the missing values with random values, which would disturb the statistical properties of the data. This would cause unexpected outputs. The Missing-indicator method is creating a dummy feature that will indicate if the values are actual observed values or they are missing and filled. Even though, missing-indicator method is commonly used in literature, it creates a bias on predictions. Filling data with mean values to preserve the statistical properties or checking other features to decide on how to fill the missing data would be the best options to avoid bias and unexpected predictions [9]. If the best practices are not applied to handle missing data, it is difficult to trust a model.

Regularization: Regularization is one of the main methods to prevent the machine learning algorithms from overfitting data. Overall regularization aims to improve generalizability of the algorithm on unseen data, as a result it increases the performance on test datasets. Regularizarion controls the fluctuation of the objective function so that the parameters of the model do not take extreme values. There are multiple possible normalization methods used such as Lasso, Ridge and ElasticNet regularization. For the training of deep neural networks with millions of parameters the use of regularization is almost a necessity to avoid memorization. Regularization is considered as a best practice in the literature and improves the model quality.

Train-Test Split: Creating train and test splits out of data is important. If the train split does not contain enough data, then the model wouldn't perform well. If the test split does not have enough data it cannot represent overall data and the performance metric values on test data cannot be generalized. Hence the split proportion must be chosen carefully. The split sizes does not only effect the performance, it also determines whether the performance metrics are still applicable. Therefore it is an important metric to consider while calculating trust score.

Factsheet Completeness: A Factsheet summarizes the most important meta information regarding one specifically trained machine learning model and they contain information regarding the structure, creator, aim of the model and the data that is used. Factsheets communicate the details of the algorithm to the end users so that end users can understand how the model is constructed and how the decisions are being made. Factsheet completeness metric measures if the factsheet includes all necessary information that the users need. It is impossible to trust a model when we are not provided with the structure of the model.

2.4.3 Limitations

The metrics explained above compose a carefully created selection. This selection covers data preprocessing, the training process, and even the communication with various stakeholders. However, the list of metrics used can still be extended.

The metrics for the training methodology pillar are not actual metrics with certain evaluation techniques. Rather they are methods to improve the quality of the machine learning algorithms. Therefore to get a score out of those metrics first we need to determine the "correct" approach and then we need to evaluate how far is the current approach to the correct one. According to this assessment, we can assign a score to the current approach considering some determined thresholds. However one must consider that each step towards getting a score out of these metrics requires a decision. The score outcome is highly related to these decisions. Decisions must have taken according to the application scenario and one should approach this issue with caution.

Chapter 3

Related Work

Trust in AI is an incipient field and no tools exist yet, that can automatically assess the trustworthiness of models in multiple dimensions. However, much work has already been done in regard to the individual pillars. The following section will shed some light on related existing work.

3.1 Related Scientific Work

Richards et al. propose the first methodology for creating factsheets as a form of AI documentation. All important stakeholders involved in the AI lifecycle, from business owners, data scientists, model validators to operations engineers contribute to the documentation. The approach helps to bridge the expertise gap between the producer and consumer of an AI service [66]. Hind et al. researched the documentation needs of developers and other stakeholders, to understand what content to include in factsheets. They highlight the importance of documenting how a model was structured, what training data was used and how features were engineered [37]. Feuerriegel, Dolata, and Schwabe introduce different notions of fairness (group fairness & individual fairness) and discuss how model developers can address the topic. Zemel et al. propose an algorithm for fair classification, that complies with the previously mentioned notions of fairness. A novel de-biasing approach called Fairnes GAN, capable of creating a releasable new dataset that plausibly approximates a given original biased dataset, was developed by Sattigeri et al. In each case, individual fairness metrics are used without combining them with each other.

In their work Tabassi et al, present a comprehensive taxonomy and terminology for adversarial robustness. The provided taxonomy includes explanations for different state-of-the-art attacks and defenses. The paper also discusses the possible consequences of the attacks and considers data integrity, confidentiality and privacy. The provided taxonomy focuses on the attacks and measuring robustness is not a concern for that work [2]. The scientific literature focuses mainly on intrinsic and post-hoc explainability methods. Post-hoc methods, like LIME, are model agnostic and focus on finding out which input features have a large impact on the output.

3.2 Related Tools

Since AI is also increasingly used in the sphere of practical IT security, both by attackers and defenders, being able to trust these tools is key. IBM's AI Fairness 360 toolkit contains a comprehensive set of fairness metrics (statistical parity, equal opportunity, average odds,..) that can be used to detect bias in machine learning models and datasets. Algorithms capable of mitigating bias during the pre-processing, in-processing and post-processing stages, cover the whole lifecylce [42] . Facebook's internal Fairness Flow toolkit is supposed to help determine whether a machine learning systems contain bias regarding specific groups of people. It was created as a reaction to the debate about responsible AI that was sparked by the Cambridge Analytica scandal [49]. Microsoft Fairlearn is an open-source project, combines interactive visualization capabilities with unfairness detection and mitigation algorithms. It is designed to help solve the sociotechnical challenge of finding a trade-off between fairness and model performance [27]. This project combines the metrics from the IBM AI Fairness 360 toolkit with metric scoring and interactive visualization. Although, AI fairness usually plays a subordinate role in IT security, the analysis can help to ensure, for example, that different types of attacks will be detected equally well.

IBM's AI Explainability 360 toolkit contains several explainability algorithms and methods [41]. The algorithms are about creating explanations for ML models or helping to make already explainable models like decision trees more accurate. It is not about evaluating the level of global model explainability but explanation methods for different types of models. It also includes two proxy metrics to evaluate the goodness of explanations. There are also various python libraries that implemented local explanations methods like LIME or SHAPE [75]. All those toolkits aim to explain the decisions of models but not to classify models on their level of explainability.

IBM's AI FactSheets 360 framework provides a guide for the preparations of factsheets. Factsheets are explanatory files that provide information regarding machine learning models such as a description of the training data, model type and information about the used training methodology. Transparent and well-documented communication between model creator and model operator increases trust and enables a more efficient integration of pretrained models. One limitation of factsheets is that their creation still requires a lot of manual effort and it would be practical to automate the process as much as possible.

Respecting robustness, the Adversarial Robustness Toolbox (ART) is an open-source Python library which is a comprehensive collection of various adversarial robustness techniques on machine learning models and it is provided by IBM. It consists of adversarial attack/defense implementations, run-time attack detection methods, poisoning detection and robustness metrics [7]. They provide a library which can be installed, imported and used during programming with Python. ART also implements multiple attack detection and defense techniques which are not in the focus of the trusted AI algorithm. Similar to the trusted AI taxonomy, ART also gathers possible robustness metrics.

Although many different libraries able to assess a single dimension of trust exist, no graphical tool aggregating several pillars was found.

3.3 Related Work in IT Security

The last decade has seen a proceeding escalation of cyber conflict. Countries began to extend their physical conflicts into the non-physical world. Cybercriminals used ransomware like WannaCry to extort millions from businesses and individuals for the decryption and recovery of their data. IT operators and cyber security professionals have to deal with a constantly changing and evolving threat landscape. The traditional IT security approach relies on deploying tools like firewalls and endpoint security mechanisms in order to enforce specific policies and provide protection against known threats. However, this only helps to a limited extent to protect against unknown threats and therefore novel self learning and adapting defense mechanisms are needed.

AI allows cybersecurity systems to become more automated and intelligent compared to the conventional security systems in the area. Sarker, Furhad, and Nowrozy provide a comprehensive overview how AI-driven cybersecurity, can leverage machine learning (ML) and deep learning (DL) methods, in order to prevent security incidents in a timely and intelligent way [70]. Unsupervised learning approaches like clustering and k-Nearest Neighbors can be used to identify hidden patterns and structures in unlabeled data. Deep learning methods are able to fulfill various purposes, ranging from intrusion and anomaly detection up to the classification of individual attacks [84]. NLP-based security modeling is able to perform advanced tasks, like vulnerability analysis of code [52].

Overall, the resultant machine learning-based security models can make intelligent cybersecurity decisions by analyzing large amounts of data from different incidents. Therefore, we can conclude that machine learning security models would be able to push the boundary of cybersecurity applications, because of their self-learning capabilities from past incidents. Several companies, like Darktrace [1] and FireEye, successfully leverage the capabilities of AI in their IT security services. Since, malicious actors may exploit the vulnerabilities of AI systems deployed by defenders, being able to validate their trustworthiness is key.

Chapter 4

Trusted AI Algorithm

The main contribution of this work is the creation of an algorithm able to quantify the trustworthiness level of machine learning models and implement and provide the algorithm with a graphical application, visualize its results, and capture them in an automatically generated trust report. In order to design such an algorithm, a taxonomy comprising the most important metrics for trust in AI was created. The theoretical background can be found in Chapter 2. Based on the identified pillars and metrics the proposed algorithm, capable of calculating the trust score for a trained ML model, was designed and implemented.

A unified way of computing the trust score for ML models allows to compare models not only with respect to their performance but also with respect to their trustworthiness. When evaluating different models, the trust score can serve as a complement to a purely performance-based assessment. Depending on the application scenario, trustworthiness can be a decisive criterion when selecting pretrained models. Tracking how the trustworthiness of a model used in operation evolves over time can help to identify possible improvements. In the following sections, the general design and structure of the trusted AI algorithm is described followed by a detailed report of the implementation of the algorithm as well as the web application it is embedded in. After that, the validation and evaluation on the basis of two application scenarios is presented. A discussion about the trusted AI algorithm's implementation and existing limitations completes this section.

4.1 Algorithm Design

Summary In this chapter, an algorithm is proposed to quantify the level of trustworthiness of ML models. The trusted AI algorithm takes into account the four pillars of trust, namely, fairness, explainability, robustness and training methodology while considering over 20 different metrics for the assessment. The general idea of the algorithm is that it takes as an input a ML model, its training/testing data, and the most important meta-information embedded in a factsheet. It then calculates all available metrics for each pillar and aggregates the results into a final trust score. The methodology and design of the proposed algorithm are described in detail below.

1 The following four dimensions of trust in AI, that have been identified during the literature review, are considered:

- Fairness
- Explainability
- Robustness
- Methodology

2 By analysing the state of the art a general taxonomy for trusted AI (Figure 4.1) was derived. It is supposed to provide an overview of all identified metrics per pillar, which are considered by the proposed algorithm.i

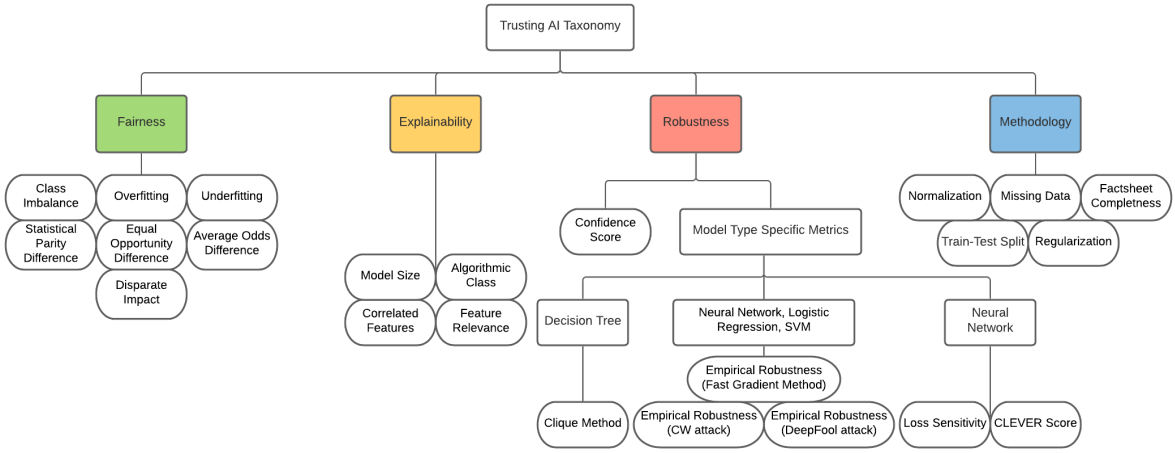


Figure 4.1: Trusted AI - General Taxonomy

3 For every metric the following four properties are defined:

- **Metric**: name to describe and reference the metric.
- **Description**: of the values which are being evaluated by the metric. For example, the used train-test split.
- **Dependency**: Lists the artefacts that the metric depends on. For the trained ML model itself, the training data, the testing data, or additional model information in the form of a factsheet.
- **Condition**: States the conditions under which a metric can be computed. For example, some robustness metrics are only applicable to certain model types, while many fairness metrics require the presence of a protected group of samples.

A detailed list of the named properties of the considered metrics per pillar can be found in the Table 4.1, 4.2, 4.3 and 4.4.

Table 4.1: Fairness Metrics Overview

Fairness			
Metric	**Description**	**Dependencies**	**Condition**
Underfitting	Difference between test accuracy and a performance baseline	Model, Test Data	-
Overfitting	Difference between the training and test performance	Model, Training Data, Test Data	-
Statistical Parity Difference	The spread between the percentage of observations from the majority group receiving a favorable outcome compared to the protected group	Training Data, Factsheet	A protected group and a favorable outcome must be defined
Equal Opportunity Difference	Difference in true positive rates between protected and unprotected group	Model, Test Data, Factsheet	A protected group and a favorable outcome must be defined
Average Odds Difference	Is the average of the difference in false-positive rates and true positive rates between the protected and unprotected group	Model, Test Data, Factsheet	A protected group and a favorable outcome must be defined
Disparate Impact	Ratio of the protected group receiving a favorable prediction divided by the ratio of the unprotected group receiving a favorable outcome	Model, Test Data, Factsheet	A protected group and a favorable outcome must be defined
Class Balance	Measure the frequency of the target classes	Training Data	-

Table 4.2: Explainability Metrics Overview

Explainability			
Metric	**Description**	**Dependencies**	**Condition**
Algorithm Class	The name of the used learning technique	Model	No combined models and learning technique must be known
Correlated Features	Percentage of highly correlated features	Training Data, Test Data	The correlation matrix of the features can always be calculated for non-missing values
Feature Relevance	Percentage of irrelevant features	Model	Must be able to calculate feature relevance scores (not possible for all model types)
Model Size	Number of features	Training Data	-

Table 4.3: Robustness Metrics Overview

Robustness			
Metric	**Description**	**Dependencies**	**Condition**
Confidence Score	Average value of the confusion matrix	Model, Test Data	Can be calculated on models which provide prediction probabilities
Clique Method	Gives a lower bound on robustness for decision tree ensembles	Model	Can be calculated on tree models
Loss Sensitivity	Quantify the smoothness of a model	Model	Can be calculated on Neural Networks
CLEVER Score	Estimates the minimal perturbation that is required to change the classification	Model	Can be calculated on Neural Networks
ER CW attack	Success rate of the CW attack	Model, Test Data	Can be calculated on Neural Networks, Logistic Regression, SVM
ER Fast Gradient Method	Success rate of the Fast Gradient Attack	Model, Test Data	Can be calculated on Neural Networks, Logistic Regression, SVM
ER DeepFool	Success rate of the DeepFool attack	Model, Test Data	Can be calculated on Neural Networks, Logistic Regression, SVM

Table 4.4: Methodology Metrics Overview

Methodology			
Metric	**Description**	**Dependencies**	**Condition**
Normalization	The name of used data normalization technique	Training Data, Test Data	-
Missing Data	Number of missing values in the Data	Training Data, Test Data	-
Regularization	The name of the used regularization technique in the training process	Factsheet	The information about whether and what regularization technique was used must be present in the factsheet
Train-Test Split	The ratio of training data to testing data	Training Data, Test Data	-
Factsheet Completeness	Percentage of required properties that must be present in the factsheet	Factsheet	-

4 In order to calculate the previously listed metrics, the following artefacts are taken as input by the algorithm:

1. ML model
2. Training data
3. Test data
4. Factsheet

For a given set of inputs, it can be derived which metrics are computable based on the given metric conditions. Every metric is implemented as a separate function, taking the above-stated inputs as parameters and returning the metric value as a result. For example, the "train-test split" metric takes as input the training and test data and returns the ratio at which the data is split into training and testing data.

5 The different metric values do not yet contain an interpretation with respect to the model trustworthiness. Therefore, the calculated metric values need to be interpreted and translated into trust scores. Every metric uses a separate function, to map the metric value to a trust score from one to five, where one corresponds to the worst score and five represents the best score.

For example, the optimal "train-test split" can be derived from the literature and currently an 80:20 split is the most frequently used. In this case, the parameters of the mapping

function would be the thresholds to map every possible split to a value from 1 to 5, where the values near the state of the art split would receive a good score and the further away the split is, the worse the score would get.

The mappings from metric value to trust score are guided as well as possible by the literature. However, mapping every possible metric value to a score from one to five with the help of thresholds or other parameters can be to some degree arbitrary. Therefore, the mapping from metric values to trust scores is parameterized and received as input in form of a $configuration_{mapping}$. A default mapping configuration is elaborated, and the trusted AI algorithm uses this if no custom configuration is provided. However, depending on the data domain or beliefs, those mapping parameters can be changed and other parameters have to be chosen to ideally map the metric value to its trust score. One static configuration cannot capture the specific requirements of every scenario. For this reason, the trused AI algorithm takes the mapping as another input parameter.

6 For a given mapping configuration, a trained ML model, its training/test data and the factsheet, a score can be calculated for every available metric. As a last step, the trusted AI algorithm aggregates all individual metric scores into a final combined value. The aggregation process is visualized in Figure 4.2, showing how the algorithm computes a final trust score from the individual metric values.

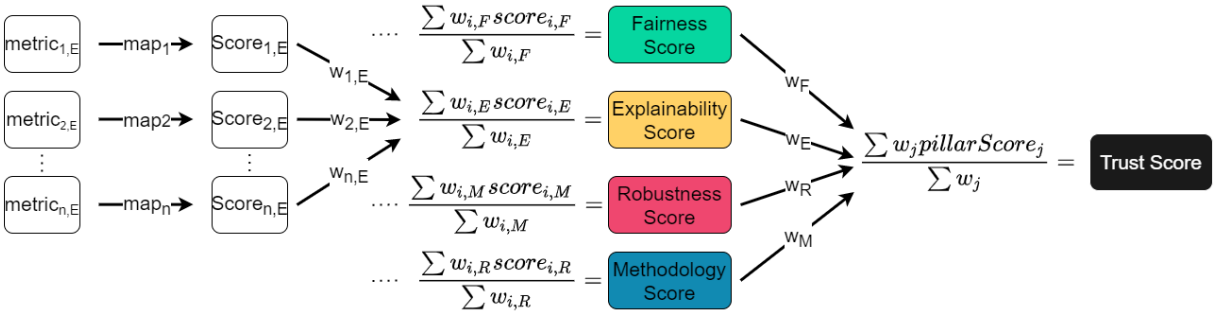


Figure 4.2: Visualization of the Trusted AI algorithm

As it can be seen, the problem is taken apart first and a trust score for every pillar is calculated. Each of the four pillars has a set of associated metrics and within that set, a weight is associated with every metric, to build a weighted average of the metric scores. The weights allow the prioritization of more important metrics. Whether all metrics are equally important and how much they should be weighted differently is up to discussion. Default weights for every metric are defined, representing their importance in the general case.

The "Algorithmic Class" is considered the most important metric for explainability, as it has a high impact on the whole working of the model and so on the level of explainability of the model. A deep neural network will most likely always be less explainable than a decision tree. Therefore, this metric receives a higher weight than other explainability metrics.

The weighted average of the metric scores is calculated for every pillar individually and thus results in an aggregated trust score for every pillar. Computing the final trust score is done analog to calculating the pillar scores. Weights are assigned to the four pillars and the final trust score is the weighted average of the pillar scores. All metric and pillar weights together are referred to as $configuration_{weights}$.

The importance of each metric and pillar is dependant on the scenario and are subjective to some degree. One might think about a scenario where humans are not making decisions based on the ML model and it is not life-threatening when the model makes bad decisions, for example, a movie recommendation system. In such a case, the explainability pillar is not as important as there is no need for the algorithm to be explainable, which could justify a weight of zero. Due to this reason, the weight of each metric and pillar is given as an optional input for the proposed algorithm. Default weights are used if no custom weights are provided.

Algorithm 1 Trusted AI Algorithm

```
1:  function TRUSTED_AI(model, data_train, data_test, factsheet, config_map, config_weights)
2:      trust_score ← 0
3:      scores_pillars ← empty dictionary
4:      for p in pillars do
5:          score_p ← 0
6:          scores_metrics ← empty dictionary
7:          metrics ← get_metrics(p)
8:          for m in metrics do
9:              args ← config_map[m]
10:             scores_metrics[m] ← get_score_m(args)
11:         for (m ∈ keys, v ∈ values) in scores_metrics do
12:             w_m ← config_weights[m]
13:             score_p ← score_p + w_m * v
14:         scores_pillars[p] ← score_p
15:     for (p ∈ keys, v ∈ values) in scores_pillars do
16:         w_p ← config_weights[p]
17:         trust_score ← trust_score + w_p * v
18:     return trust_score
```

Algorithm 1 describes the process behind the trusted AI algorithm. The trained machine learning model, its training/test data, its factsheet, the $configuration_{mapping}$ and $configuration_{weights}$ are received as input. The algorithm processes every pillar separately (line 4) and calculates the pillar scores, which are then saved in a map (line 14). For this, the metric scores are calculated using the corresponding mapping configuration (line 8). The metric scores are then aggregated into a pillar score with the weights taken from the configuration (line 11). Finally, the four pillar scores are aggregated into a final trust score (line 15), which is the return value of the algorithm. The algorithm can also be adapted to give as an output not only the final trust score but a map of all computed trust scores (metric scores, pillar scores and overall trust score).

4.2 Deployment

To validate the suitability and demonstrate the usefulness of the proposed algorithm, it got implemented as a proof of concept and was deployed as a web application. The implemented application allows end-users to analyze and compare machine learning models on their trustworthiness. The analysis is based on the implemented version of the trusted AI algorithm using the standard configuration.

This chapter explains the implementation details of the algorithm and the web application. The suitability of the proposed web application and algorithm is demonstrated and validated in the following two scenarios: IT Security (IoT Attack Classification) and Credit Card Approval. Each scenario has considered various solutions consisting of different ML models, data and training methodologies. Finally, a discussion is provided about the strength and weaknesses of the implementation and how it could be improved in the future. The code can be found on GitHub [46].

4.2.1 Algorithm Implementation

The trusted AI algorithm was implemented in Python since the most prominent machine learning libraries are implemented in Python as well. This approach makes it possible to quickly develop models to test the algorithm and its implementation.

The implemented solution supports models created with the popular ML libraries: Scikit-learn, TensorFlow and Keras, but does not support PyTorch yet. The trained models can be persisted in the respective formats shown in Table 4.5.

Table 4.5: Supported libraries and model persistence formats

ML Library	Support	Model Persistence Formats
Sklearn	✓	Pickle (.pickle) & Joblib (.joblib)
TensorFlow	✓	HDF5 (.h5) & SavedModel (.pb)
Keras	✓	HDF5 (.h5) & SavedModel (.pb)
PyTorch	×	Pickle (.pth) & ONNX (.onnx)

The inputs of the implemented trusted AI algorithm and their data types are:

- Model (Object)
- Training data (DataFrame)
- Testing data (DataFrame)
- Factsheet (Dictionary)
- Weights Configuration (Dictionary)
- Mapping Configuration (Dictionary)

The model is a fitted ML model object from one of the supported ML libraries. This allows to call its functions and inspect its hyperparameters, which is important for the calculation of certain metrics. The factsheet, the mapping configuration, and the weights' configuration are given as dictionaries with a specific structure. The default weight configuration is shown in Listing 4.1, which defines a weight for every metric and every pillar.

Listing 4.1: Default Configuration of the Weights

```
{
    "fairness": {
        "underfitting": 0.35,
        "overfitting": 0.15,
        "statistical_parity_difference": 0.15,
        "equal_opportunity_difference": 0.2,
        "average_odds_difference": 0.1,
        "disparate_impact": 0.1,
        "class_balance": 0.1
    },
    "explainability": {
        "algorithm_class": 0.55,
        "correlated_features": 0.15,
        "model_size": 0.15,
        "feature_relevance": 0.15
    },
    "robustness": {
        "confidence_score": 0.2,
        "clique_method": 0.2,
        "loss_sensitivity": 0.2,
        "clever_score": 0.2,
        "er_fast_gradient_attack": 0.2,
        "er_carlini_wagner_attack": 0.2,
        "er_deepfool_attack": 0.2
    },
    "methodology": {
        "normalization": 0.2,
        "missing_data": 0.2,
        "regularization": 0.2,
        "train_test_split": 0.2,
        "factsheet_completeness": 0.2
    },
    "pillars": {
        "fairness": 0.25,
        "explainability": 0.25,
        "robustness": 0.25,
        "methodology": 0.25
    }
}
```

The weights are used to calculate the weighted average for the aggregation of the metric scores to pillar scores and the aggregation of the pillar scores to the overall trust score. The weights reflect the individual importance of metrics and pillars. In the default configuration, the pillars are equally weighted, meaning they all contribute equally to the overall trust score.

It can be seen that the metric "algorithm_class" is allocated a weight of 0.55 (Line 11). This is because it is the most important explainability metric since it has a high impact on the whole working of the model. For example, a deep neural network will probably always be less explainable than a decision tree. The other three explainability metrics received an equal weight of 0.15 as there is no clear way to tell if one of the metrics is slightly more important than the other in the general case. In the fairness pillar, the metric "underfitting" (Line 3) receives the highest weight, because if the model is underfitting the model is not good and should not be trusted. The robustness and methodology metrics are all equally important and therefore receive equal weight.

The implementation of the algorithm is split into five python files, which can be can found in the *algorithm* folder [80]. The file structure follows the algorithm design presented in Figure 4.2. There is one file per pillar to calculate the metric scores for that pillar and another file called "trustworthiness" to aggregate the metric scores.

Function 1 General Metric Function

1: **function** GET_METRIC_XY($model, data_{train}, data_{test}, factsheet, config_{map}$)
2: $Mapping\_Parameters \leftarrow config_{map}[XY]$
3: $Metric\_Value \leftarrow Get\_Value\_XY(model, data_{train}, data_{test}, factsheet)$
4: $Metric\_Score \leftarrow Get\_Score\_XY(Metric\_Value, Mapping\_Parameters)$
5: **return** $Metric\_Score,\ Metric\_Value$

The metric functions in the pillar files calculate the metric scores. If a metric is not computable due to unmet preconditions, then the metric score is of type *NaN* (not a number). Every metric has its function called *Get_Metric_metric_name* (Function 1), which all have the same general structure. They do not only return the score but also the calculated metric value, like the train-test split for example. The function takes as an argument the model, training/testing data, factsheet and mapping configuration. The function then computes the metric value with the *Get_Value* function taking the given inputs. The metric value is then mapped to a score of 1-5 with the *Get_Score* function, taking the parameters associated with that metric, from the mapping configuration, as input. Finally, the function returns the metric score but also the metric value, to extract the trust-related property as well.

For the mapping configuration, there are default values defined for every parameter and a description of how they are used is provided. This can be found in the configuration folder [81]. Some of the metric functions are presented in detail now:

Underfitting (Fairness)

This metric takes as input the model, the test data and the thresholds parameter, which is [0.8, 0.85, 0.9, 0.95] by default. The metric value is the accuracy of the model evaluated on the test data. This value can easily be calculated by calling the *fit* function of the model on the test data. To map the test accuracy to a score from 1-5 the threshold parameters are used which create 5 bins. If the accuracy is below the first number (0.8 by default) the mapping would result in the worst score, which is one. if the test accuracy is higher than the last number (0.95 by default) a score of five would be achieved. The mapping is done analog to the thresholds in between.

Overfitting (Fairness)

This metric takes as input the model, the test data, the train data and the thresholds parameter, which is [0.04, 0.03, 0.02, 0.01, 0] by default. The metric value is the difference between the training and the test accuracy of the model. This value can easily be calculated by calling the *fit* function of the model on the training and test data and taking the difference of those results. With this value, overfitting can be detected, namely if the training accuracy is significantly higher than the test accuracy. To map the value to a score from 1-5 the threshold parameters are used which create 5 bins. If the training accuracy is over 4 percentage points (first threshold) higher than the test accuracy, that would result in a score of 1. But if the difference is zero (last threshold) or below, meaning the test accuracy is equal to or higher than the training accuracy, the mapping would result in a score of 5. The mapping is done analog to the thresholds in between.

Statistical Parity Difference (Fairness)

The value of this metric is the statistical parity difference and its value function is presented in Function 2. At first, the data is split into a protected and an unprotected group. This is done based on the specific value of the protected feature. Afterward, for both groups, the ratio of samples receiving a favorable outcome is computed. The absolute difference of these two ratios is then called the statistical parity difference. This value is then mapped to a score with the thresholds parameter which is [0.075, 0.05, 0.025, 0.01, 0] by default. Since the ideal value would be 0, meaning no difference between the two groups, the thresholds are created such that a value of 0 would result in a score of five. Everything difference higher than 0.075 results in a score of 1.

Function 2 Statistical Parity Difference - Metric Value Function

```
1:  function GET_VALUE_STATISTICAL_PARITY_DIFFERENCE(model, data, factsheet)
2:      protected_feature ← factsheet["protected_feature"]
3:      protected_values ← factsheet["protected_values"]
4:
5:      protected_group ← filter(data[protected_feature].is_in(protected_values))
6:      unprotected_group ← filter(data[protected_feature].not_in(protected_values))
7:
8:      protected_group_size ← size(protected_group)
9:      unprotected_group_size ← size(unprotected_group)
10:
11:     target_column ← factsheet["target_column"]
12:     favorable_outcomes ← factsheet["favorable_outcomes"]
13:
14:     favored_protected_group ← filter(protected_group[target_column].is_in(favorable_outcomes))
15:     favored_unprotected_group ← filter(unprotected_group[target_column].is_in(favorable_outcomes))
16:
17:     favored_protected_group_size ← size(favored_protected_group)
18:     favored_unprotected_group_size ← size(favored_unprotected_group)
19:
20:     favored_protected_ratio ← favored_protected_group_size/protected_group_size
21:     favored_unprotected_ratio ← favored_unprotected_group_size/unprotected_group_size
22:
23:     statistical_parity_difference ← |favored_unprotected_ratio − favored_protected_ratio|
24:     return statistical_parity_difference
```

Equal Opportunity Difference (Fairness)

The computation of the equal opportunity difference is presented in the Function 3. First, the set of samples is split into a protected and unprotected group. The split is done based on the values of the protected feature. For both groups, the ratio of favorable samples receiving a favorable prediction is calculated. The absolute difference between these two ratios is called equal opportunity difference. The default thresholds parameter is: [0.15, 0.1, 0.05, 0.03, 0], with which the equal opportunity difference is mapped to a trust score. A difference of 0 would result in a score of 5 and a difference higher than 0.15 in a score of 1.

Function 3 Equal Opportunity Difference Metric - Metric Value Function

```
function GET_VALUE_EQUAL_OPPORTUNITY_DIFFERENCE(model, data, factsheet)
    protected_feature ← factsheet["protected_feature"]
    protected_values ← factsheet["protected_values"]

    protected_group ← filter(data[protected_feature].is_in(protected_values))
    unprotected_group ← filter(data[protected_feature].not_in(protected_values))

    target_column ← factsheet["target_column"]
    favorable_outcomes ← factsheet["favorable_outcomes"]

    favored_protected_group ← filter(protected_group[target_column].is_in(favorable_outcomes))
    favored_unprotected_group ← filter(unprotected_group[target_column].is_in(favorable_outcomes))

    favored_protected_group_size ← size(favored_protected_group)
    favored_unprotected_group_size ← size(favored_unprotected_group)

    predicted_favored_protected_group ← filter(favored_protected_group[y_pred].is_in(favorable_outcomes))
    predicted_favored_unprotected_group ← filter(favored_unprotected_group[y_pred].is_in(favorable_outcomes))

    predicted_favored_protected_group_size ← size(predicted_favored_protected_group)
    predicted_favored_unprotected_group_size ← size(predicted_favored_unprotected_group)

    predicted_favored_protected_ratio ← predicted_favored_protected_group_size/favored_protected_group_size
    predicted_favored_unprotected_ratio ← predicted_favored_unprotected_group_size/favored_unprotected_group_size

    equal_opportunity_difference ← |predicted_favored_protected_ratio − predicted_favored_unprotected_ratio|
    return equal_opportunity_difference
```

Algorithm Class (Explainability)

This is the most important metric from the explainability metric. Its value is in contrast to most other metrics not a number but a string, name of the class of the ML Model. This metric has as input only the model and the algorithm class can be extracted by inspecting the python object which represents the ML Model. The names are specific to the used python ML libraries.

To get a trust score out of the algorithm class name, the names have to be mapped to scores from 1-5. This is done with the mapping parameter which is a simple dictionary (see the default mapping in Listing 4.2). The mapping is inspired by the literature. The level of explainability each algorithm provides was extensively reasoned about in Subsection 2.2.2. It can be seen from the default mapping that using a decision tree model would result in a score of 5 and a neural network (MLPClassifier) in a score of 1.

Listing 4.2: Algorithm Score Mapping

```
{
    "RandomForestClassifier": 3.5,
    "KNeighborsClassifier": 3,
    "SVC": 2,
    "GaussianProcessClassifier": 3,
    "DecisionTreeClassifier": 5,
    "MLPClassifier": 1,
    "AdaBoostClassifier": 3,
    "GaussianNB": 3.5,
    "QuadraticDiscriminantAnalysis": 3,
    "LogisticRegression": 4,
    "LinearRegression": 4,
    "Sequential": 1
}
```

Correlated Features (Explainability)

The metric value is the percentage of highly correlated features. It takes as an input the training and test data, the thresholds parameter, which is [0.05, 0.15, 0.25, 0.4] by default, and an additional parameter called *high_correlation_threshold*, which is 0.9 by default. This additional parameter is used to define what a high correlation means. The default value of 0.9 means that every correlation equal to or higher than 0.9 is considered a high correlation. The Function 4 describes that metric, where first the correlation of every feature is calculated. Then, the algorithm checks if the correlation is over the given threshold and counts every occurrence of a high correlation. In the end, the ratio of highly correlated features to the total number of features is returned. The ratio is then translated into a trust score from 1-5 with the given thresholds. As it can be seen from the default parameter if below 5% of the features are highly correlated the metric score would be 5.

Function 4 Correlated Features - Metric Value Function

function GET_VALUE_CORRELATED_FEATURES($test\_data, train\_data, high\_corr\_threshold$)
    $data \leftarrow concat(test\_data, train\_data)$
    $correlation\_matrix \leftarrow correlation(data)$
    $upper\_matrix \leftarrow upper\_triangle(correlation\_matrix)$

    # Count occurrences of high correlation

    $count\_high\_corr \leftarrow 0$
    # Check for every correlation between features if they are over the threshold
    for $corr \in upper\_matrix$ **do**
        if $corr > high\_corr\_threshold$ **then**
            $count\_high\_corr \leftarrow count\_high\_corr + 1$
    $pct\_corr \leftarrow \frac{count\_high\_corr}{numer\_of\_features(data)}$
    return pct_corr

Empirical Robustness (Robustness)

The calculation of the three empirical robustness metrics follows the same flow (see Function 5). First, a subset of the test data is randomly sampled because as the amount of the test data increases the calculation for the attacks slows down. To keep the function fast enough, a smaller subset of the test data is being used. Later the accuracy of the small test data is measured. Using the same test data, the attacks are applied and adversarial examples are created. In this step, the existing attack implementations from the ART library are used. Next, the accuracy score on the adversarial examples is measured. The empirical robustness score depends on the normalized ratio of the difference between before attack, after attack accuracy.

To map the difference between before-after attack accuracy to a trust score of 1-5, the array [80, 60, 40, 20] is used as the default thresholds parameter. If the before attack accuracy is 85% and after attack accuracy is again 85% it means the model is so robust that attack was not successful at all. The difference is 0 and hence this would result in a score of 5 (best score). However, if the before attack accuracy was 100% and after attack accuracy is 0% then the model is not robust. The difference is 100% and it would result in a score of 1 (worst score).

Function 5 Empirical Robustness Metric - Metric Score Function

function GET_SCORE_EMPIRICAL_ROBUSTNESS($model, test\_data, thresholds$)
    # Randomly sample test_data to get a small subset
    $sample\_X, sample\_y \leftarrow sample(test\_data)$

    # Predict and calculate before attack accuracy
    $first\_prediction \leftarrow model.predict(sample\_X)$
    $before\_attack\_accuracy \leftarrow accuracy\_score(sample\_y, first\_prediction)$

    # Calculate attack and create adversarial examples on the sampled test data
    # Here ART library attack implementations are used
    # The inputs to the attack function vary according to the attack type
    $adversarial\_example \leftarrow ART\_attack(model, sample\_X, sample\_y)$

    # Predict and calculate after attack accuracy
    $second\_prediction \leftarrow model.predict(adversarial\_example)$
    $after\_attack\_accuracy \leftarrow accuracy\_score(sample\_y, second\_prediction)$

    $success\_ratio \leftarrow \frac{before\_attack\_accuracy - after\_attack\_accuracy}{before\_attack\_accuracy} \times 100$

    # Assign score according to the thresholds
    $score \leftarrow calculate\_score(success\_ratio, thresholds)$
    return *score*

Confidence Score (Robustness)

The confidence score is computed using the normalized confusion matrix (see Function 6). The diagonal of the confusion matrix represents the confidence in the correct predictions. The score is based on average confidence. The higher the confidence the better and for the default thresholds parameter the array [60, 70, 80, 90] were chosen, wherewith the confidence of over 90% a score of 5 is reached.

Function 6 Confidence Score - Metric Score Function

function GET_SCORE_CONFIDENCE_SCORE($model, test\_data, thresholds$)
    # Calculate confusion matrix
    $prediction \leftarrow model.predict(test\_data)$
    $confusion\_matrix \leftarrow confusion\_matrix\_function(test\_data, prediction)$
    $normalized\_confusion\_matrix \leftarrow \frac{confusion\_matrix}{rowwise\_sum(confusion\_matrix)}$

    # Calculate confidence score using confusion matrix
    # Only use diagonal because diagonal shows confidence on correct predictions
    $confidence \leftarrow normalized\_confusion\_matrix.predict$
    $average\_confidence \leftarrow average(confidence)$

    # Assign score according to the thresholds
    $score \leftarrow calculate\_score(average\_confidence, thresholds)$
    return *score*

Train-Test Split (Methodology)

The metric value function of the train-test split takes as input the training and test data. The split can easily be computed by comparing the size of the two data sets. A well-known state-of-the-art split is dividing the data into 80% training data and 20% test data. This split would receive a trust score of 5 and the further away from the split from the ideal value is the worse the score will get. The exact default mapping is shown in Listing 4.3.

Listing 4.3: Score Mapping

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

Factsheet Completeness (Methodology)

This metric is checking the entries of the factsheet. If the factsheet has an entry in all 8 categories a score of 5 is achieved. For every missing entry, 0.5 points are subtracted from the score. The final number is transformed into an integer with a floor function to get the metric score.

Aggregation Process

Next to the metric functions, every pillar file also contains the main function called *Analyze* (see Function 7), which calls all the metric functions of that pillar and saves the results. The function creates and returns two dictionaries one for the metric scores (Scores_MAP) and the other for the metric values (Values_MAP). The dictionaries have as keys the metric names and as value the metric scores or the metric values respectively. Those pillar maps are then combined together to create a single dictionary including all metrics. In Listing 4.4 an example of such a dictionary is provided and it can be seen that some metrics (Line 20-24) could not be computed. This is done for scores and values by the *combine* function from the "trustworthiness" file, which simply calls the *Analyze* functions from the pillar files and combines the results.

Function 7 General Pillar Function

1: **function** ANALYZE($model, data_{train}, data_{test}, factsheet, config_{map}$)
2: $Input \leftarrow model, data_{train}, data_{test}, factsheet, config_{map}$
3: $Scores\_MAP \leftarrow empty\ dictionary$
4: $Values\_MAP \leftarrow empty\ dictionary$
5: **for** $m\ in\ metrics$ **do**
6: $Metric\_Score,\ Metric\_Value \leftarrow Get\_Metric_m(Input)$
7: $Scores\_MAP[m] \leftarrow Metric\_Score$
8: $Values\_MAP[m] \leftarrow Metric\_Value$
9: **return** $Scores\_MAP,\ Values\_MAP$

Listing 4.4: Scores Dictionary (Example)

```
1  {
2      "fairness": {
3          "underfitting": 3,
4          "overfitting": NaN,
5          "statistical_parity_difference": 1,
6          "equal_opportunity_difference": 1,
7          "average_odds_difference": 1,
8          "disparate_impact": 5,
9          "class_balance": 5
10     },
11     "explainability": {
12         "algorithm_class": 4,
13         "correlated_features": 2,
14         "model_size": 2,
15         "feature_relevance": 5
16     },
17     "robustness": {
18         "confidence_score": 5,
19         "clique_method": 3,
20         "loss_sensitivity": NaN,
21         "clever_score": NaN,
```

```
        "er_fast_gradient_attack": NaN,
        "er_carlini_wagner_attack": NaN,
        "er_deepfool_attack": NaN
    },
    "methodology": {
        "normalization": 1,
        "missing_data": 5,
        "regularization": 1,
        "train_test_split": 5,
        "factsheet_completeness": 5
    }
}
```

In the ”trustworthiness” file there are two other functions, which aggregate the scores. The function *get_pillar_scores* takes as input the dictionary with all calculated metric scores (like Listing 4.4) and a weights configuration (like Listing 4.1). For every pillar, it calculates the weighted average of the available metrics. To save the result a dictionary with the pillars as keys and the weighted scores as values is created. If for the scores the example from Listing 4.4 and for weights the default configuration (Listing 4.1) is chosen, then the function output would result in a dictionary-like it is seen in Listing 4.5.

Listing 4.5: Pillar Scores (Example)

```
{
    "fairness": 2.5,
    "explainability": 3.6,
    "robustness": 4,
    "methodology": 4.5
}
```

The final function called *get_final_score* takes these pillar scores and the weights configuration as input and calculates the weighted average of the pillar scores, which is the final trust score. This final score is then returned. To get the final trust score the three functions from the ”trustworthiness” file need to be called in sequence.

The trusted AI algorithm is therefore split across multiple functions, which allows to not only get a final score but also to get all the intermediate scores and values as well. The algorithm is implemented in a very generic way such that it is easy to extend the set of metrics or add a new pillar.

4.2.2 Web Application

For the implementation of the web application, the python framework dash [61] is used. That framework is chosen because the algorithm is also written in python and could therefore be easily integrated into the application without the need for an API and input transformations. Furthermore, the flexible modular framework enables to be efficient many iterations are possible, where new ideas can be gathered and better functionality and usability can be added to achieve a good final version of the web application.

To structure the database and web application better, two new terms are defined:

- Scenario: Defined as a generic ML task with a clearly defined goal and an unprocessed data set. For example, the classification of different cyberattacks attacks based on collected Raspberry Pi system data from normal and infected devices.
- Solution: A ML model solving the task of the scenario is then referred to as a solution. For one scenario there can be multiple solutions, which consists of a fitted ML model, the processed training/testing data and a factsheet.

At the core of the trusted AI application is the algorithm. The frontend of the application allows users to interact with the algorithm and view the trust analysis for given ML models. Figure 4.3 gives an overview of the architecture of the application. There are two types of modules namely the *Upload Modules*, with which the user can create a scenario and upload a ML solution for a certain scenario to the server, and the *Analyse Modules*, with which the user can get the trust analysis, calculated with the trusted AI algorithm, for a given solution and is able to compare different solutions with each other. The analysis is displayed graphically and can be interactively configured and explored to a high level of detail. The backend hosts the proposed Trusted AI Algorithm and a database in charge of saving the ML models and configurations.

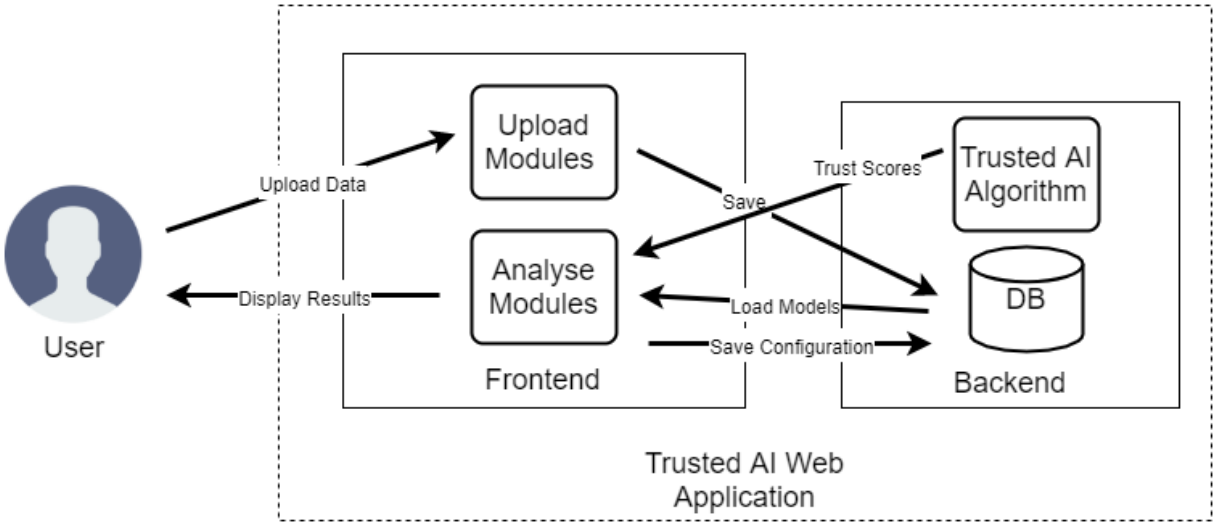


Figure 4.3: The architecture of the Trusted AI Web Application

The application is structured in 4 pages: The *Scenarios* and *Upload* page (upload modules) and the *Analyze* and *Compare* page (analyze modules). This can be seen in Figure 4.4, the main navigation bar, which also reflects the flow of the application and how it is meant to be used.

Figure 4.4: Navigation

On the first page, the user can create and define a scenario or explore already saved scenarios. On the next page, the user can upload the solutions for the defined scenario and provide additional information about the solutions. Then on the analyze page the user can view the trust analysis for a given solution and adapt the algorithm configuration to the scenario. The user also can download the full trust analysis for the solution as a PDF. On the last page, the user can compare the trust scores of different solutions within a scenario with each other.

Scenario Page

All saved scenarios are displayed on that page. Per the available scenarios, the following information is shown: The name, a description of the scenario, the link to the data set and the names of the uploaded solutions for that scenario. The user also can add new scenarios (Functionality 1), which will then be available for selection on the other pages.

Functionality 1 Adding a Scenario

Input

– Scenario Name – Description – Link to data set (optional)

Trigger

Press the plus button on the right corner

Action

Saving the scenario to the server and include it in the set of available scenarios.

Upload Page

On the upload page, the user can add the ML models and give additional important information in the form (Figure 4.5). The upload process is formally described in Functionality 2. First, a scenario has to be selected and a name must be given to the solution. After that, the training and testing data can be uploaded, which can be either a *csv* file or a *pickle* file. Once the data is uploaded a preview of the data is shown to crosscheck the data.

Functionality 2 Upload Data of the Model

Input

- Scenario: Select one of the saved scenarios
- Solution: Name of the solution
- Description: Brief description of the solution (optional)
- Training data: Drag and drop file (file format: *csv* or *pickle*)
- Testing data: Drag and drop file (file format: *csv* or *pickle*)
- Protected Feature: Features like age, race or gender (optional)
- Protected values: Protected values within the protected features (optional)
- Target Column: Column that contains the values to be predict with the model
- Factsheet: Drag and drop or create (file format: *json*)
- ML model: Drag and drop fitted model from scikit-learn [60] or tensorflow [32]

Trigger

Press the ”Analyze” button

Action

Saving the solution with the model and data to the server and saving all other information to the database. After the data is successfully uploaded the user will be redirected to the analyze page.

Functionality 3 Creating and Saving a Factsheet

Input

- Model Name (optional)
- Purpose: Describes the purpose of the model (optional)
- Domain: Describes the domain the model is intended to be used (optional)
- Data: A description of the data and the preprocessing (optional)
- Model information: Most important information about the model (optional)
- Authors: Who created the model (optional)
- Contact information: Contact in case of questions about the model (optional)
- Regularization: Select the type of regularization used (default: None)

Trigger

Press the ”Create Factsheet” button

Action

From the inputs a factsheet in the form of a structured json file will be created. It can be directly used in the upload page (Functionality 2) to save it to the database or downloaded for later use (storing metadata information).

Also, a factsheet about the model has to be provided. The application proposes two different approaches, in the first one a prepared factsheet can be uploaded as a *json* file. The second approach is to create and save the factsheet using the web interface. When clicked on the "Create Factsheet" button a new form appears (Figure4.6), which asks general information about the model such as "Who developed the model and what is the contact information of the developer?" or "What is the name of the model and what is its purpose?" (see Functionality 3).

The information from the factsheet is important when it comes to accountability and transparency but it is not required to fill out. Whenever this form is complete the user can save the input to the database or download the created factsheet for later use. Finally, the trained ML model must be uploaded. By clicking the "analyze" button all data from the input form will be saved to the database and the user is redirected to the analyze page.

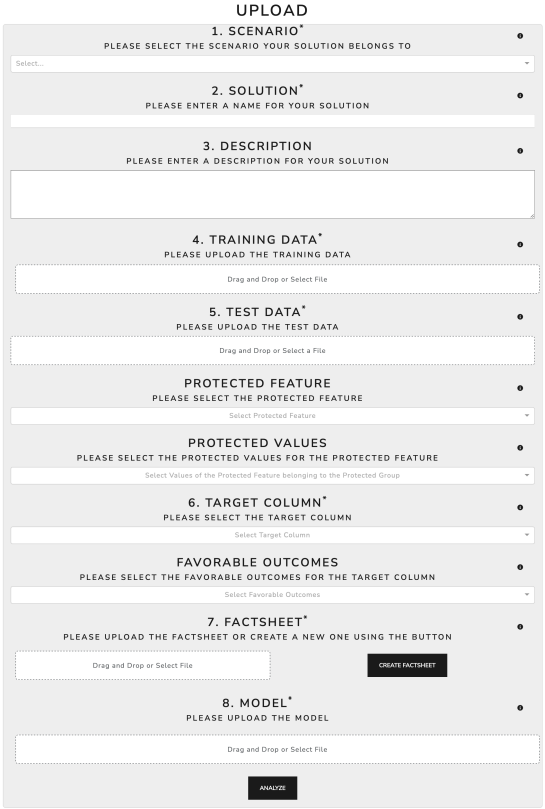


Figure 4.5: Upload

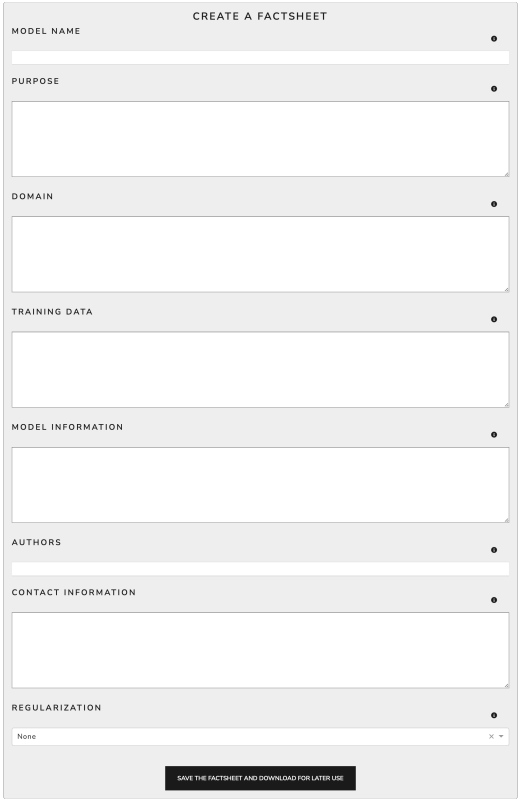


Figure 4.6: Factsheet

All the optional information provided in the upload page like the factsheet or the description of the model is used for the creation of the automated trust report by the analyze page or to display it as general information in the analyze modules. Some of the information, like the protected features in the data set or the completeness of the factsheet, is also used in the calculation of certain metric scores by the trusted AI algorithm.

Analyze Page

The main page of the application is the analyze page. There, a user gets the trust analysis of his/her solutions. A scenario and a solution from that scenario have to be selected. Once a solution is selected, the trust scores will be automatically computed and the interactive report of the trust analysis will be displayed (Functionality 4).

Functionality 4 Display the Interactive Trusted AI Report

Input

- Scenario: Dropdown of the saved scenarios
- Solution: Dropdown of all solutions for the chosen scenario

Trigger

Callback on the solution dropdown (automatically triggered when a solution gets selected or changed)

Action

Run the trusted AI algorithm with the model, training/testing data and factsheet as input taken from the database for the selected solution. Save the results of the algorithm in cache and compute performance metrics on the testing data with the given model. Use those results together with the additionally saved data for that solution to create an interactive trust report.

The trust report consists of two parts: A general information section, which provides an overview of the chosen solution, and a trustworthiness section, which shows the result of the trusted AI algorithm with an interactive top-down approach, that first gives an overview of the overall trust score and the pillar scores and allows to go further into detail on the individual pillars. The mapping and weights configuration is also shown and can be adapted.

The report starts with the general information section (Figure 4.7) and consists of several parts. There is a short description of the scenario and a general description. This information is from the database and was once provided earlier in the upload modules.

Below on the left, there is a short table of performance metrics, which were calculated on the fly by evaluating the fitted ML model of the selected solution on its testing data. Even though the focus is not on the performance metrics they are still important to get an overview of the model.

The calculated performance metrics are:

- Accuracy
- Global Recall
- Class Weighted Recall
- Global Precision
- Class Weighted Precision
- Global F1 Score
- Class Weighted F1 Score

In the bottom right corner, there is a short summary of the most important trust-related metric values (properties). They help the user to get a quick overview of the information extracted by the trusted AI algorithm.

The selected properties are:

- Model Type (Algorithm)
- Train Test Split
- Train Test Size
- Regularization Technique
- Normalization Technique
- Number of Features

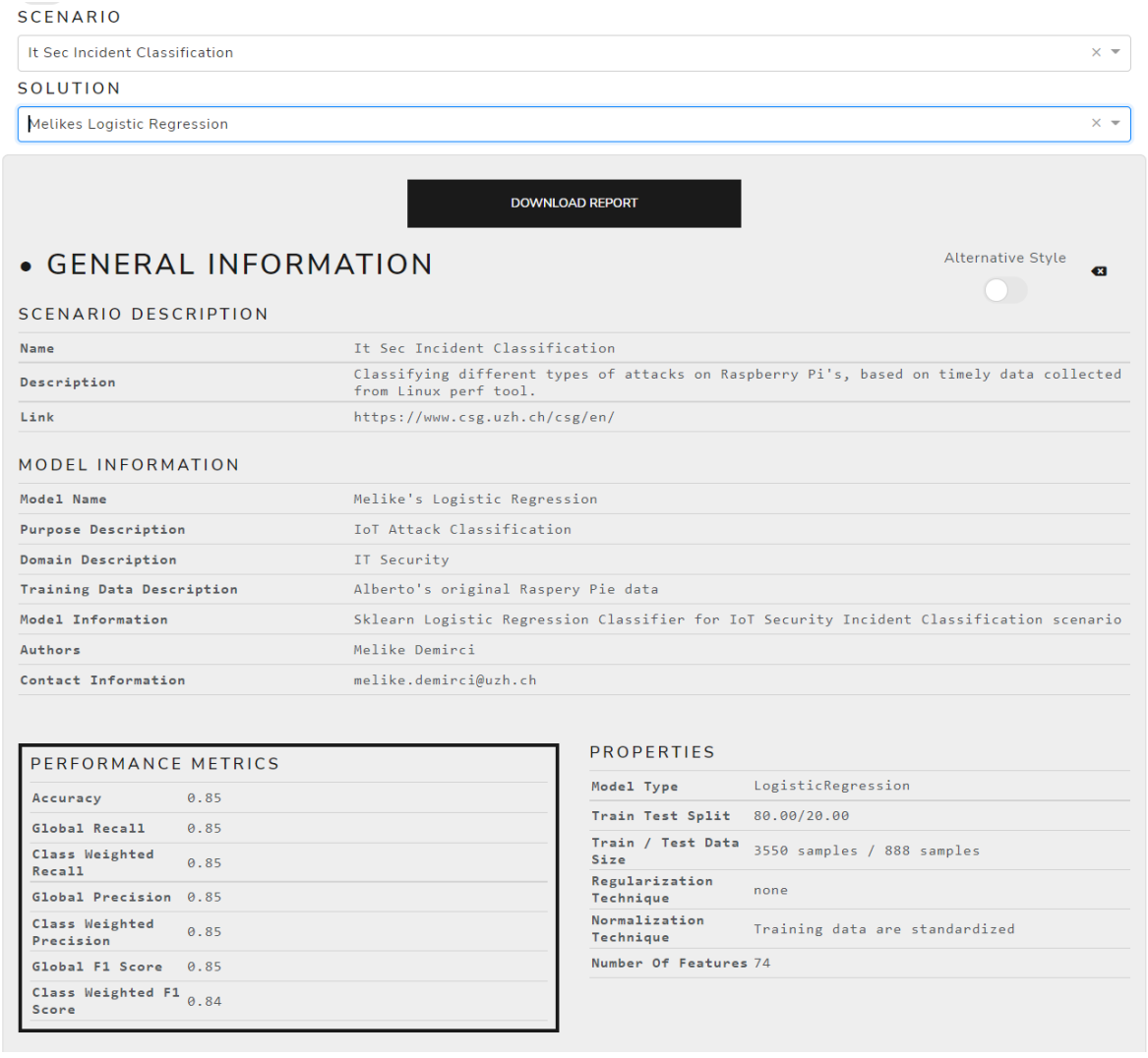


Figure 4.7: General Information Section

The second part of the report is focused on the visualization of the results of the trusted AI algorithm. The user is presented first with an overview (Figure 4.8) where pillar scores and the overall trust score are shown. Looking at the overview the user already knows the overall level of trustworthiness of the selected solution and which pillar has a strong or weak trust score.

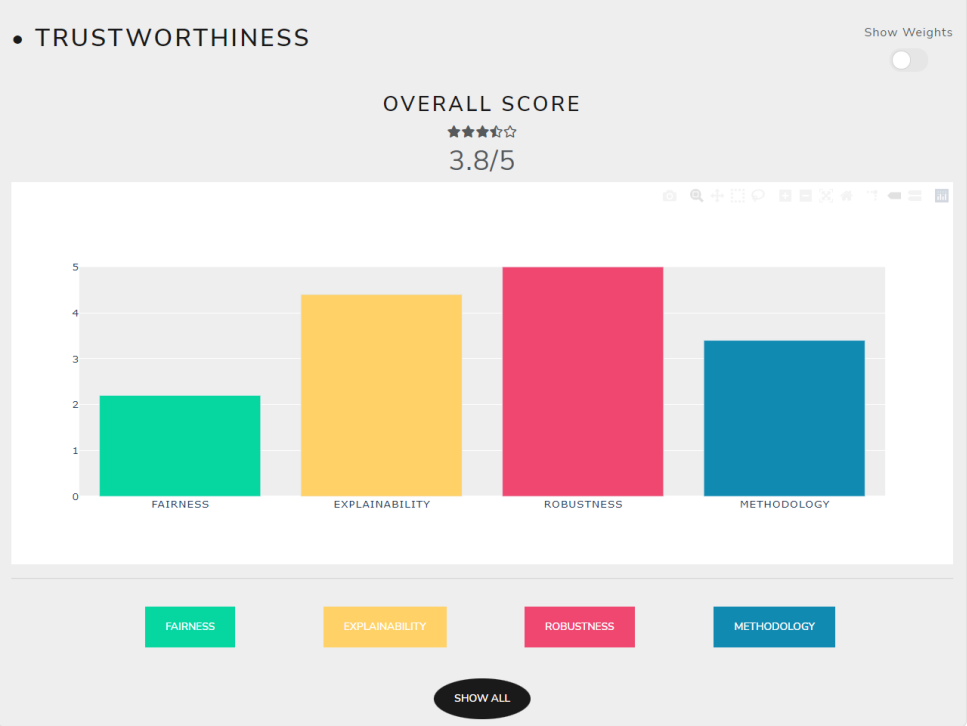


Figure 4.8: Overall Trust score and Pillar Scores

From this first overview, the user can click on one of the buttons below each of the pillars to get a more detailed view of the metric scores of that pillar. It is also possible to go one layer further and check out the individual metric values and what they depend on.

As it can be seen in Figure 4.8, in the right corner, the user can enable the ”show weights” button, which will open the weights configuration (Figure 4.9). The weights are used to aggregate the individual metric scores to pillar scores and then aggregate the pillar score to an overall trust score. Therefore the weight affects the pillar scores and the overall trust score but not the metric scores. The weights can be seen as the importance of the pillars and metrics. If all weights would be the same it would mean that all metrics and all pillars are equally important.

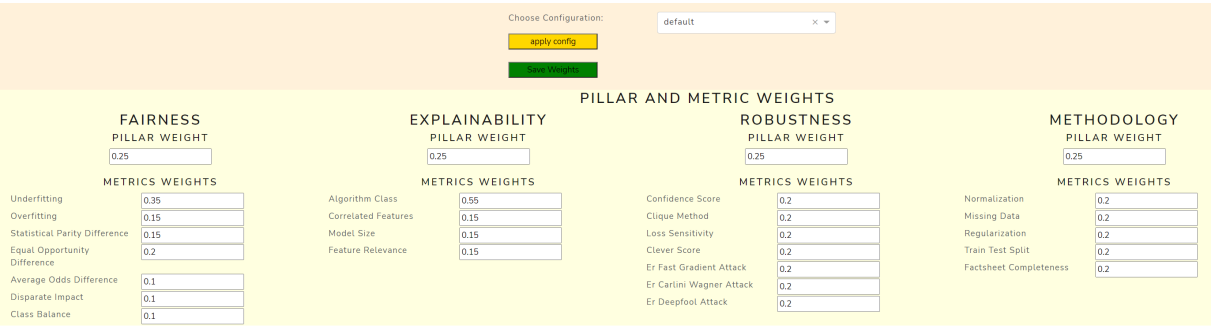


Figure 4.9: Weights Configuration

At first, the weights are always set to the default configuration setting, which can also be seen in Figure 4.9. With the change of those weights, the user can enter his own views about the importance of metrics and pillars for the scenario or solution. The weights configuration can be saved (Functionality 5) or previously saved configurations can be loaded through the dropdown menu. With the button ”apply config” the current weight values will be taken for the weights configuration and the aggregation of the metric and pillar scores will be recalculated. The effect of the changed configuration can be immediately seen in the chart from Figure 4.8.

Functionality 5 Save Weights Configuration

Input

- All pillar and metric weight values
- Name of Configuration

Trigger

Click Button ”save weights”

Action

The weight values will be put together to a *json* file and saved to the database under the given input name. The new weights configuration will be immediately available for selection.

As mentioned earlier and seen in the overview chart in Figure 4.8. Below each of the pillars is a button with which the user has the possibility to get a more detailed view of that pillar, by displaying the pillar section. An example of how the pillar section looks can be seen in Figure 4.10, which shows a plot of the metric scores of the fairness pillar. It visualizes all computable metrics of the selected solution and if you hover over the metric a short description of the metric is displayed. Here the user already gets a better insight into each metric and sees directly which metrics scored high or low.

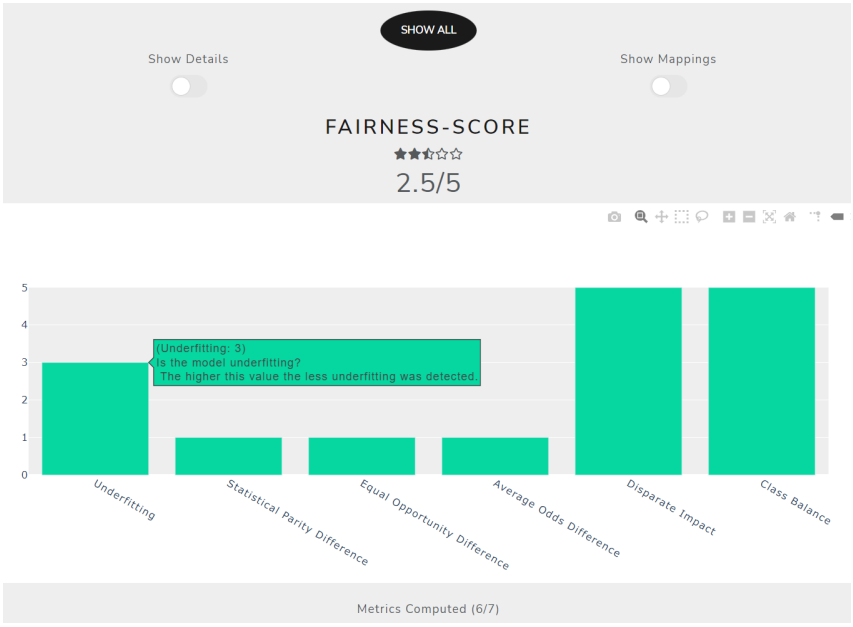


Figure 4.10: Metric Scores - Fairness Pillar

At the bottom of the chart, it is also indicated how many of the totally available metrics for that pillar could be computed and are therefore visible in the chart. On the top-left corner of the pillar section, the user can click on the button ”show detail” to open the metric detail section and get even more information about the individual metrics.

The detail section not only shows the metric score but also the calculated values of the metrics for example the evaluated train-test split or the number of features the model uses. For every metric, there is a small description of the metric, the metric values and a list of the inputs its computation depends on (model, training data, testing data or factsheet). Figure 4.11 shows a part of the detail section of the fairness pillar. In the detail section, all available metrics can be found and for those which can not be computed the reason is provided. This level of detail makes the trust report transparent.

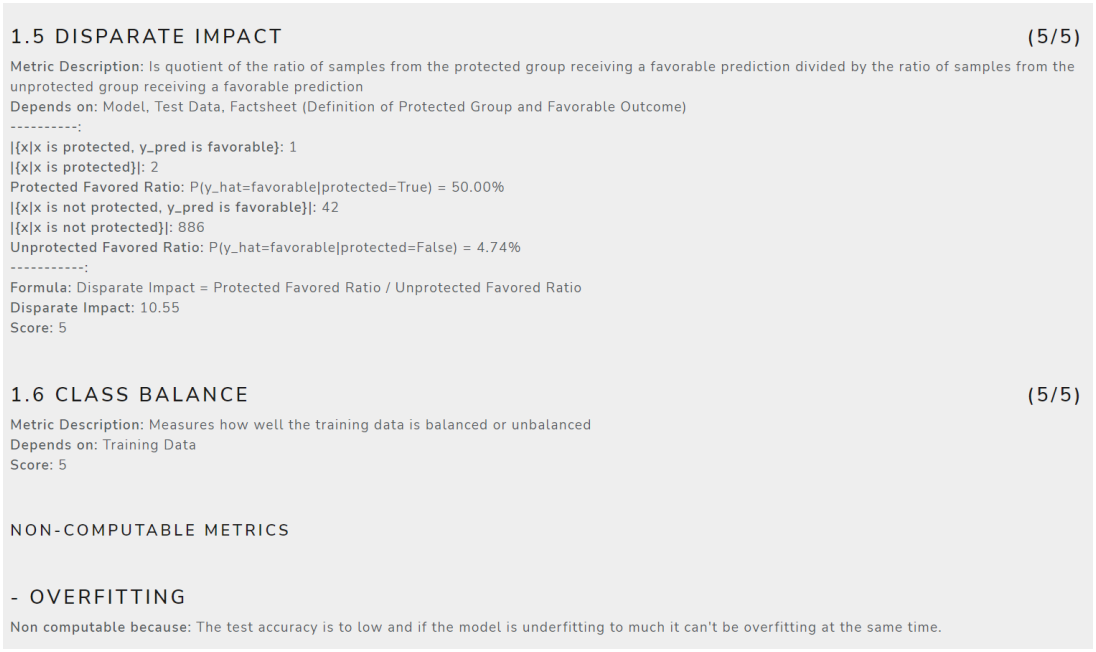


Figure 4.11: Metric Detail Section - Fairness Pillar

It can be seen in Figure 4.10 that in the pillar section there is not only the button on the left to see the metric details but also another button in the top right corner, which will open the mapping configuration. A part of the default mapping configuration of the explainability pillar is seen in Figure 4.12. It shows the parameters for the mapping functions. Those are the parameters that are used to map metric values to metric scores. Every parameter can be changed and those changes can be saved (Functionality 6) to the database and applied (Functionality 8) or an already saved mapping configuration can be loaded (Functionality 7). For every mapping parameter, there is a description showing up when hovering over it, which instructs the user on how the parameters (mainly thresholds) map the value to a trust score.

For example if in Figure 4.12 the user hovers over the input form of the metric ”correlated features” the following text would show up:

”Thresholds of how to map the percentage of highly correlated features to a score from 1-5. For example if 0.05 is the first number it means that training data set with 5% or less highly correlated variables would get the best score(5) and if 0.4 was the last number it would mean that a training data set 40% or more highly correlated features would get the worst score (1), analog for the numbers in between.

For that metric the user can set the threshold for the classification of a high correlation, which is set to 0.9 by default, and how to map the percentage of highly correlated features to a trust score. Those thresholds can be tailored to the beliefs of the user. Because of that mapping configuration, it is transparent for every metric how the trust score for a metric is computed from the metric value.

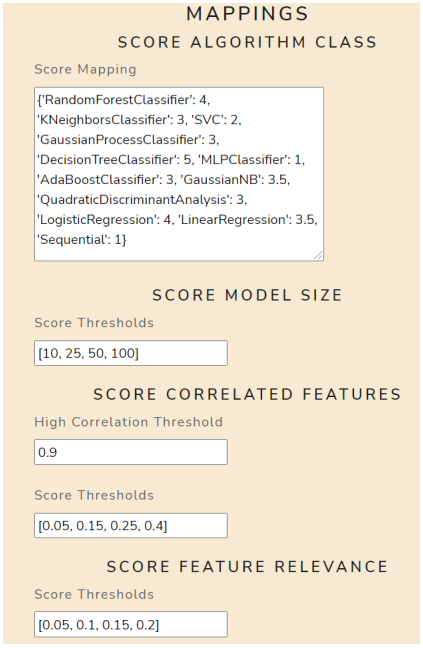


Figure 4.12: Mapping configuration - Explainability Pillar

Functionality 6 Save Mapping Configuration

Input

- All inputs from the form (mapping parameters)
- Name of Configuration

Trigger

Click Button ”save”

Action

The values from the mapping input form will be put together to a *json* file and saved to the database under the given input name. The new mapping configuration will be immediately available for selection.

Functionality 7 Load Mapping Configuration

Input
Configuration Name (selected from the dropdown menu of all previously saved mapping configurations)

Trigger
Callback on the configuration dropdown menu

Action
The saved mapping configuration for that pillar will be loaded from the database and will be entered as the input values of the form.

Functionality 8 Apply Mapping

Input
None

Trigger
Click Button ”apply”

Action
The current values from the configuration input form will be put together to a *json* file and will become the new mapping configuration. The trusted AI algorithm is run again with this new mapping configuration as input.

On the analyze page the user is presented with a graphical representation of the results of the trusted AI algorithm for a single solution. General information about the solution is shown as well and a short overview table of the most important performance metrics is provided. The user can get a detailed view of each of the pillars where all calculated metric values and metric descriptions and dependencies are shown.

The trusted AI algorithm uses a weights configuration that represents the importance of each metric and pillar and is used for the aggregation of the metric scores. Also for every metric there exist mapping parameters (mostly thresholds), which are used to go from a metric value to a metric trust score. This mapping configuration and the weights configuration can both be changed and saved by the user. As it can be seen in Figure 4.7 at the top the user has the possibility to download a PDF report (Appendix A), which comprises all information on the analyze page for a given solution and configuration. This automated trust report is not only showing the trust scores but also all the metric values. It could therefore also be viewed as an automated factsheet.

Compare Page

The last page is the comparison page (Figure 4.13). For a given scenario the user can compare the trust scores of two different solutions within that scenario with each other (Functionality 9). The user can also choose the configurations to be applied for the two solutions. The page displays a compact side-by-side view of the trust report of the solutions. For each solution, a short description of the solution and the performance metrics, the chart of the pillar trust scores (like Figure 4.8) and the charts of the metric scores for every pillar (like Figure 4.10) are shown.

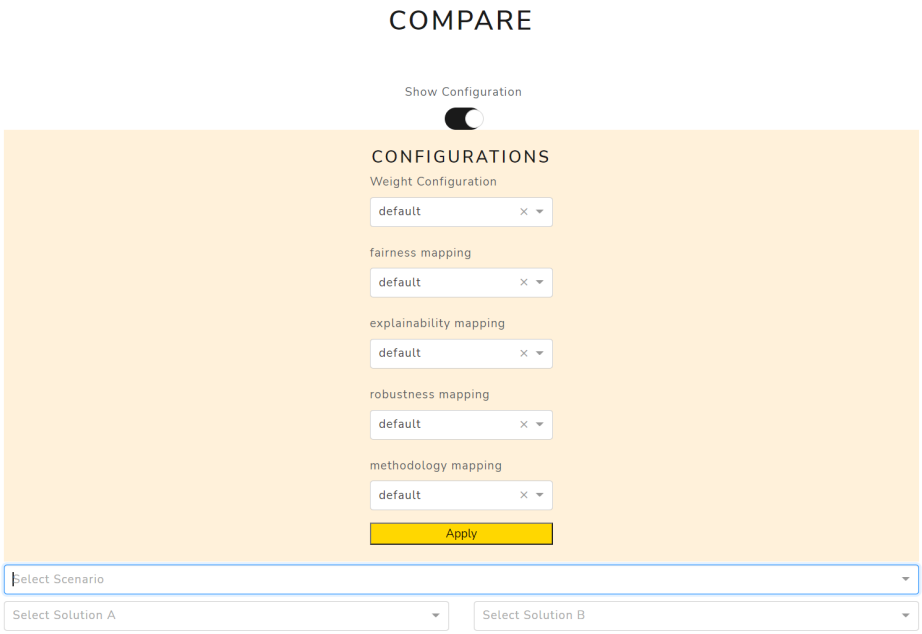


Figure 4.13: Compare page

Functionality 9 Compare Solutions

Input

- Scenario
- Solution A (a saved solution from the selected Scenario)
- Solution B (another saved solution from the selected Scenario)
- Weights Configurations
- Mapping Configuration form all pillars

Trigger

Callback on the solution inputs (automatically triggered when a solution gets selected or changed

Action

A compact version of the trust report of the two solutions will be presented side by side.

4.3 Evaluation

To accurately evaluate the trusted AI algorithm and its implementation, two different use scenarios were chosen. The first one is IoT Attack Classification from IT Security which is an analytical application that does not involve a lot of social constructs. The second scenario is a Credit Card Approval process which is deeply concerned with fairness and explainability aspects. For each scenario, multiple machine learning solutions were created. For the validation, the theoretical expectations were compared against the empirical results observed in the application.

IT Security - IoT Attack Classification

The first scenario used in the validation of the trusted AI algorithm is from the IT Security field specifically on IoT Attack Classification. This problem is focused on detecting different cyberattacks affecting a Raspberry Pi 3. The dataset models the internal behavior of the Raspberry Pi related to the network, CPU, memory, tasks, the amount of executed instructions, cache fails, etc. The data has been collected from the Linux perf tool both when the Raspberry Pi is up-and-running normally and when it is infected with 6 different attacks. There are around 4450 data samples with 73 features. The classification task is to evaluate whether the Raspberry Pi is running normally or if not what the attack type is.

Solutions for IoT Attack Classification: The machine learning models trained for a specific scenario are referred to as solutions. Various solutions were created for the validation of the trusted AI algorithm. We have compared the individual pillar values and overall trust score both for different types of machine learning algorithms used and also the same algorithms trained using different hyperparameters.

The solutions created for the IoT Attack Classification scenario include DT, RFC, LR, SVC which are all trained using sklearn implementation, and also Keras Neural Network Classifiers.

To validate the trusted AI algorithm and the web application, theoretically expected results were determined to be compared against the real outcomes of the application. One of the theoretical expectations was to see high explainability scores on Decision Trees and low explainability scores on Neural Networks. The observed results in the trusted AI web application were in line with the theoretically predicted outcome.

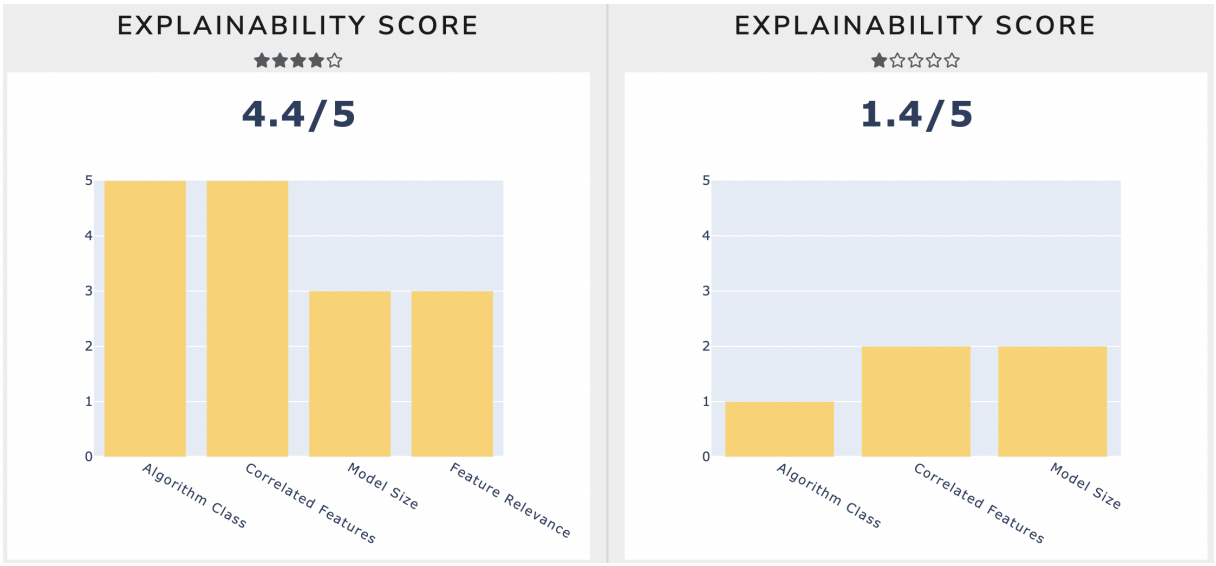


Figure 4.14: Explainability Scores of a DT (left), NN (right)

The created Decision Tree Solution had a 4.4/5 explainability score and the neural network had 1.4. There is an immense difference especially in the Algorithm Class which is not surprising. During the creation of the Decision Tree, highly correlated features have been removed which leads to a higher score from the Correlated Features metric and also a higher score from Model Size since the number of features decreased. The alignment between the monitored outcomes and the theory validates the trusted AI algorithm and the implementation.

In terms of robustness, it is expected for a simple Logistic Regression Classifier solution to have a significantly low robustness score. Especially the attacks that are chosen for ER metric are known to have more than 95 percent success rate on Logistic Regression from the theoretical background this would result in low ER scores. The output of the trusted AI algorithm to a Logistic Regression model is shown in Figure 4.15. All ER attacks had high success ratios on the model which lead to low scores. Since the logical conclusions and empirical results are in line with each other we can once again confirm our implementation.

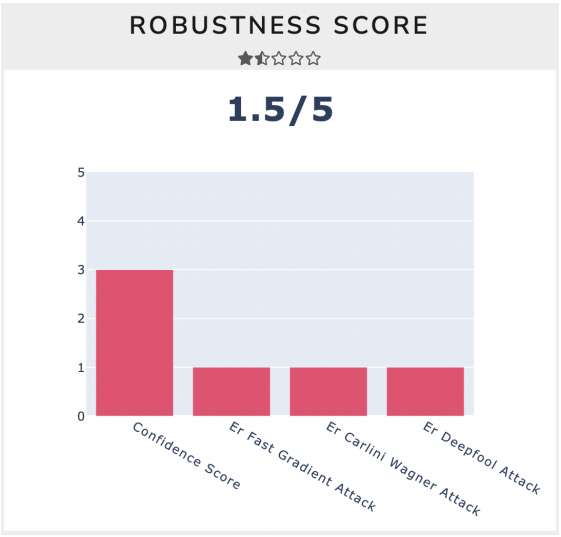
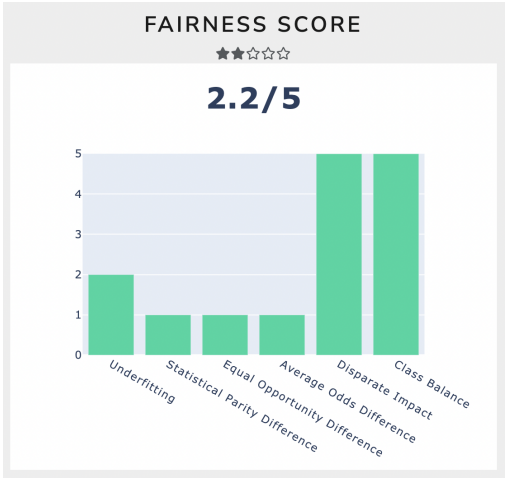
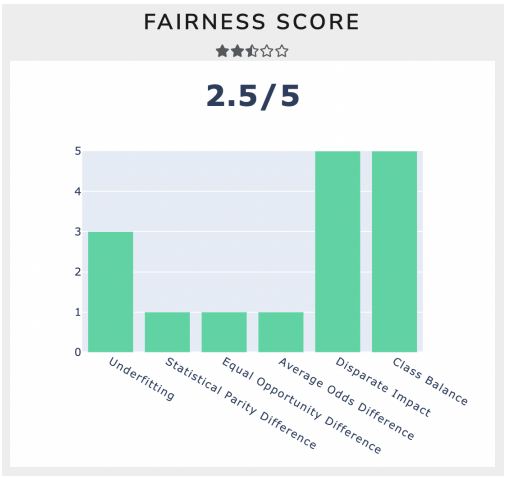


Figure 4.15: Robustness Score of LR solution

A comparison of one DT and RFC models' fairness scores is provided in Figure 4.16. The models are trained with the same data on a scenario that is not related to fairness. Hence only metric that performs differently is the underfitting. DT slightly overfits because it is a simpler model, but RFC fits better.



DT fairness score



RFC fairness score

Figure 4.16: Performance Metrics of SVCs

To compare the Trust Scores and to visualize the misleading effect of the classical performance metrics, two SVCs have been trained using the same dataset. For this purpose, the "Compare" page on the trusted AI web application was used to view the results of different solutions side by side.

The performance metrics calculated for the two SVCs can be found in Figure 4.17. At first glance, SVC1 seems like a superior model as it performs better. However, our research proposes that the evaluation of machine learning algorithms solely depending on performance metrics can be misleading.

PERFORMANCE METRICS	
Accuracy	0.87
Global Recall	0.87
Class Weighted Recall	0.87
Global Precision	0.87
Class Weighted Precision	0.88
Global F1 Score	0.87
Class Weighted F1 Score	0.88

SVC1

PERFORMANCE METRICS	
Accuracy	0.85
Global Recall	0.85
Class Weighted Recall	0.85
Global Precision	0.85
Class Weighted Precision	0.85
Global F1 Score	0.85
Class Weighted F1 Score	0.85

SVC2

Figure 4.17: Performance Metrics of SVCs

To compare to different solutions accurately one should consider their trustworthiness as well. Checking the trust scores of the SVCs reveals how misleading the performance metrics could be. As the Figure 4.18 shows, SVC1's trust score, on the left, is lower than SVC2's.

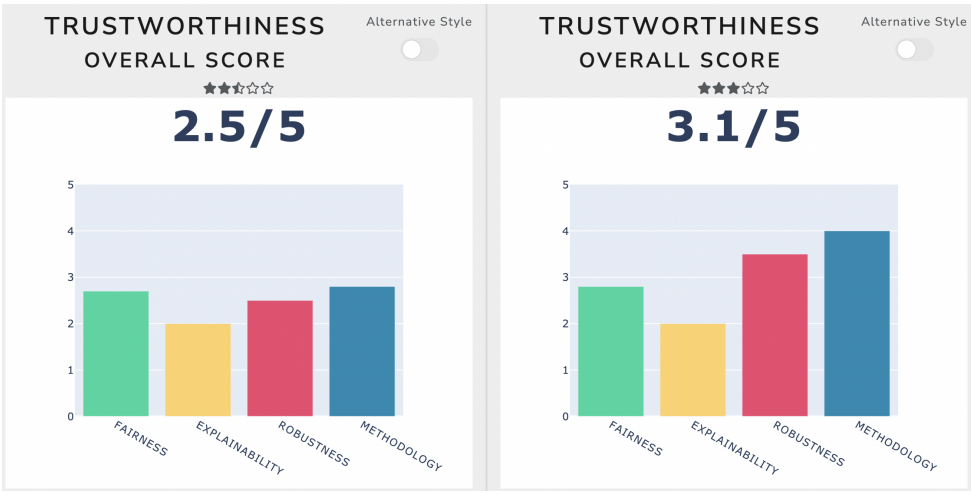


Figure 4.18: SVC Trustworthiness Comparison

This result shows SVC2 is actually superior in terms of trustworthiness, it is more robust and has a higher Training Methodology score and is slightly better in terms of fairness. This example confirms the need for the trust score and it shows how classical performance metrics are not enough while comparing machine learning algorithms.

Credit Card Approval

In theory and practice, many examples exist where machine learning models for credit scoring have shown discriminatory behavior towards protected groups. Therefore, the approval of credit cards was chosen as the second scenario. The task is to decide, based on data about the applicant, whether to accept or reject the application for a credit card. The scenario is based on a synthetic Kaggle dataset [20]. For the calculation of the fairness metrics, it is crucial to divide the population into a protected minority and an unprotected majority group. For this purpose, certain values of a protected attribute are selected which define the protected minority. In the case of the first scenario, there was no intuitively sensitive attribute that should not be used for classification. Therefore, the second scenario was added to complement the validation.

Solutions for Credit Card Approval Solutions created for the credit card approval scenario include SVM's, KNN's, RFC and Hinge classifiers with stochastic gradient descent learning. The Hinge classifier was trained using the SGDClassifier implementation in sklearn[73] and it will be referred to as SGD from now on.

As mentioned before this scenario is concerned with fairness and explainability aspects and is used for the validation of those pillars. For all solutions, the same source of data

with class imbalance was used. For example, Figure 4.19 shows a comparison of fairness scores of a SGD model and a KNN. Since KNN is a very simple model we expect that it does not fit the data well. Also, we would expect that SGD would handle the imbalance data better and generalize better compared to the KNN.

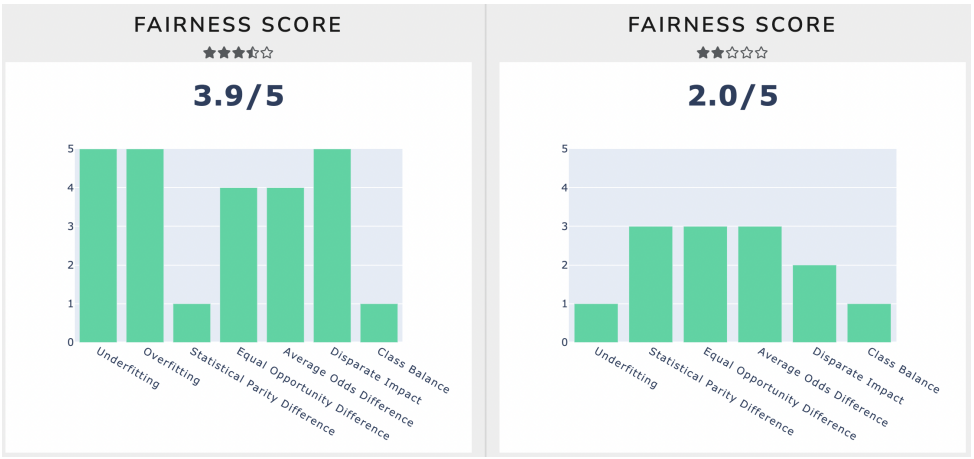


Figure 4.19: Fairness comparison between SGD (on the left) and KNN on the right

As Figure 4.19 shows, the fairness score of the KNN is significantly lower than the SGD. As predicted KNN model underfits. In contrast, SGD has a high underfitting and overfitting value, which implies that the model fits well. Also, other metrics prove that the effects of the data imbalance did not reflect on the outputs of SGD as much as they did on KNN's.

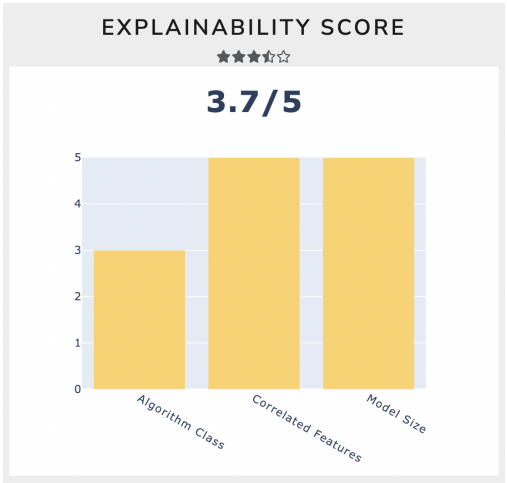


Figure 4.20: Explainability of a KNN

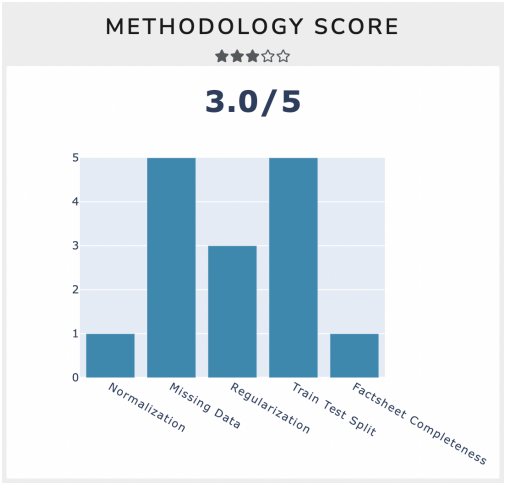


Figure 4.21: Methodology of a KNN

Figure 4.20 shows the explainability score of a KNN model. KNN's are known to be relatively simple and understandable, therefore we expect to observe a high explainability score. As it can be seen in the figure, the model gets a score of 5 both in Correlated Features and Model Size metrics, resulting in an overall high explainability score. The score for the Algorithm Class metric is 3 which shows that the evaluated model is simple.

Figure 4.21 shows the methodology score of the same KNN model. Here one can see that the train-test split is inlined with the best practices and the missing data points are handled well both in train and test datasets. The Regularization metric score is 3 which means during the training regularization was used. Low Factsheet Completeness score means that all important data regarding the training and evaluation of the model is not provided by the factsheet. A low Normalization metric score implies that there was no normalization used on the train or test data.

Figure 4.22 shows the robustness score of a SVM model. Comparing the results to Figure 4.15 one can see how a SVM model performs better in terms of robustness compared to a much simpler LR model. SVM gets a robustness score of 5 and it gets higher scores from all of the ER metrics.

Overall looking at the visualizations provided by the trusted AI web application, it can be seen that the application provides good insights through each of the pillars regarding the model details. Each metric represents decisive information regarding the trustworthiness of the model.

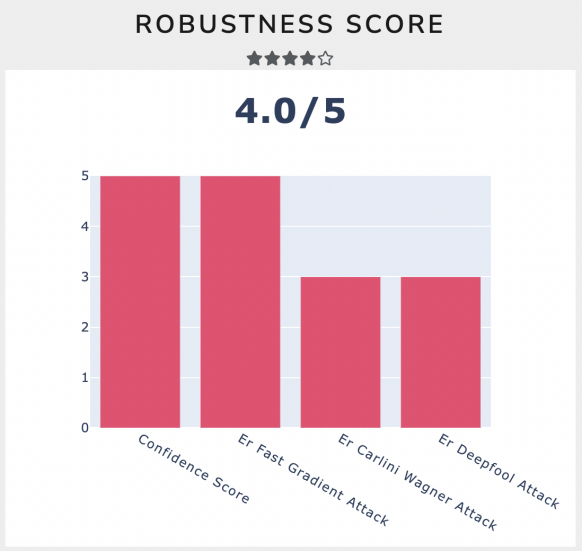


Figure 4.22: Robustness Score of SVM solution

Chapter 5

Discussion

5.1 Contribution

Taxonomy
By analyzing the scientific literature, the main four pillars of trust in AI were identified. With intense literature research and looking at the state-of-the-art ML methodologies, a total of 22 important trust metrics, belonging to one of the four pillars, were elaborated. With those metrics and pillars, a general taxonomy for trusted AI. Such a taxonomy of metrics unifying different dimensions of trust has not been created yet. The taxonomy combines the knowledge from different pillars and provides and an overview of computable trust-related metrics.

Algorithm Design
The created taxonomy provided the foundation for the trusted AI algorithm. It declares the logic of weighting metrics and mapping the metric values to trust scores for 1-5, followed by aggregating those metric scores to pillar scores and finally to trust scores. The design is modular and metric weights and the mappings from metric values to metric scores are given as inputs to the algorithm. The algorithm can easily be extended with more pillars and metrics.

Algorithm Implementation
The implemented algorithm is capable of handling trained models from scikit-learn and TensorFlow. The hierarchical structure of the implemented trusted AI algorithm allows one to not only get a trust score in the end but to get trust scores in every step, which makes the algorithm very transparent. It is possible to examine the level of trustworthiness for each pillar and with respect to every single metric. Adjustable parameters, like the metric weights or mappings from weights to scores, are given as input to the algorithm. This flexibility allows to adapt the algorithm to the specifics of the scenario. The modular design of the algorithm allows to easily extend it in the future with further pillars and metrics.

Validation
To validate the suitability and demonstrate the usefulness of the proposed algorithm, it was deployed within a Python-based web application. The implemented application allows end-users to analyze and compare machine learning model levels of trust. It further allows to semi-automatically generate factsheets and extend them with trust-related properties. The analysis generated by the trusted AI algorithm can be downloaded as a compact PDF report, containing all computed trust scores, metric values and general model related information. The implemented solution was validated using two application scenarios, which confirmed the usability of the algorithm and provided detailed insights into the trustworthiness of the different solutions with good visualization and detailed descriptions of each of the metrics. The application enables the user to add new scenarios and solutions and to fully tailor the parameters to any application scenario .

5.2 Justification of the Taxonomy

As seen in Chapter 2 the taxonomy, including its pillars and metrics is backed up by scientific literature. Every pillar is essential for providing trust-related information. However, the taxonomy was created for the scope of classification and is therefore missing dedicated metrics for regression models. Further dimensions of trust in AI like *privacy* or *data security* can be found in the literature. The taxonomy could be extended with additional pillars and corresponding metrics. The existing four pillars, could be extended with additional metrics.

5.3 Justification of the Algorithm

Since the algorithm somehow tries to classify algorithms into trustworthy or non-trustworthy, we tried to analyze the trusted AI algorithm itself on its level of trustworthiness. We consider the algorithm to be *fair* since every assumption is parameterized and given as an input. If one does not agree on the prioritization of metrics or pillars or the thresholds to compute trust scores one can simply use another mapping or weights configuration.

The algorithm is also *explainable*. The argumentation is very similar to that of a decision tree, one can just follow the branches and see which metric scores had a positive or negative impact on the final score. This is very useful as low scores can be spotted right away and it is known where the metrics would have to improve to receive a higher trust score. The calculation of the individual metric trust scores is also transparent since the mapping parameters and thresholds are known, meaning one can tell exactly how a certain metric value had to change to receive a higher score.

The algorithm seems to be fair and explainable, but is it also *robust*? Since the weights and mapping parameters are given as an input, a very small change in the inputs can lead to a big change in the output. For example, all weights except for one metric and the corresponding pillar could be set to 0 and the thresholds could be constructed such that a small change in the metric value could result in changing a score of 1 to a score of

5. Our algorithm is in that case not robust and strongly dependent on the configuration. But also with the default configuration the output of the algorithm could change a lot with small changes, for example changing the learning technique of the model could result in a very different score for the explainability pillar and some metrics in the robustness pillar might not be computable anymore and therefor might change the score drastically as well. However, these were extreme cases to show strong aberrations are possible, but in the normal case, a small change in the input has almost no effect on the final score at all since there are over twenty metrics that get aggregated and the change in the input must be strong enough to surpass the threshold to receive a new score.

At the core of the algorithm lies the trusted AI taxonomy. To get to this taxonomy the *methodology* was reading up on trustworthiness in AI and searching the literature for different available metrics of trust. The categorization of trust in the four pillars is disputable and other dimensions of trust could also be considered like data privacy. Also, the list of metrics could most probably be extended and more metrics could be added to the algorithm. However, all of the metrics are backed up by the literature and are relevant for the level of trust and therefore must be considered in the algorithm.

To wrap it up, overall we evaluated the trusted AI algorithm as trustworthy. It is able to quantify the trustworthiness of machine learning classification models. The standardized trust score can be used to compare it against other models and be integrated into the model selection process and model monitoring. The mapping and weights configuration can be tailored to any scenario to give back the most accurate trust score.

5.4 Limitations

So far the implemented trusted AI algorithm can only take classification models, build with scikit-learn or TensorFlow, as input. In the future, support for more machine learning libraries could be added. The implementation could also be extended in order to support regression models, by adding dedicated metrics. Unsupervised learning approaches have not yet been implemented nor validated by the algorithm yet. The set of metrics would have to be revised and possibly extended. Most implemented fairness and robustness metrics can not be used for unsupervised learning. However, the methodology and explainability metrics do not have this restriction. Federate learning models are also more difficult to validate with the current implementation, as the training data and the training methodology is normally not accessible. The lack of this information prohibits the computation of most fairness and methodology metrics. If multiple training data sets and methodologies are used (e.g., in case of federated machine learning), each set has to be evaluated separately and then aggregated into a final score. This could also be implemented as part of future work.

Chapter 6

Summary and Conclusions

In a first step, an understanding of the existing pillars (fairness, robustness, explainability, and accountability) was established. This provides relevant insights to detect trusted AI models and understand their decisions. Limitations and drawbacks of individual metrics were also addressed. Our findings were condensed into one general taxonomy of pillars, features and metrics relevant for trusted AI. Two application scenarios were used in order to validate the practical applicability of the taxonomy. The trusted AI algorithm was implemented in an online platform, which is able to calculate individual scores per pillar and combine all theses scores into a global one. The application scenarios were used in order to evaluate our implementation in terms of technical details, performance, and usability. Good results were achieved in all these dimensions. Our project helps to bridge the trust and knowledge gap between model developers and model consumers.

6.1 Future Work

So far, our tool supports machine learning models used for classification. In the future, support for regression models should be added. Improvements could be made in the structure of the application and in the calculation of the metrics, for example by improving the computation speed. Furthermore, when adjusting the settings of a metric, it should be automatically recalculated and updated. Currently, only the performance of a model is displayed. The application could make concrete suggestions to the user on how to improve the model. For example, bias mitigation algorithms could be suggested. The space that a metric occupies in the overview should be inversely proportional to the achieved result. Currently, metrics with good results take up a lot of space and vice versa. In the future, however, the user's attention should be directed to the aspects that perform less well and need improvement. This could be achieved by inverting the scores making 1 the best score and 5 the worst. Since there is a lot of current research regarding trust in artificial intelligence, the state of the art will progress and new methodologies and tools are going to be developed. Therefore, it is definitely an interesting field to work in. Maybe a scientific publication containing the results of this project might be produced.

Bibliography

[1] Darktrace AI. "Combining Unsupervised and Supervised Machine Learning". In: (2021).

[2] Elham Tabassi et al. "A Taxonomy and Terminology of Adversarial Machine Learning". In: (2019). URL: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8269-draft.pdf.

[3] Fuxun Yu et al. "Interpreting and Evaluating Neural Network Robustness". In: (2019). URL: https://www.ijcai.org/proceedings/2019/0583.pdf.

[4] Hongge Chen et al. "Robustness Verification of Tree-based Models". In: (2019). URL: https://arxiv.org/abs/1906.03849.

[5] Ian Goodfellow et al. "Explaining and Harnessing Adversarial Examples". In: (2014). URL: https://www.researchgate.net/publication/269935591_Explaining_and_Harnessing_Adversarial_Examples.

[6] Kevin Eykholt et al. "Robust Physical-World Attacks on Deep Learning Visual Classification". In: (2018). URL: https://arxiv.org/pdf/1707.08945.pdf.

[7] Maria-Irina Nicolae et al. "Adversarial Robustness Toolbox v1.0.0". In: (2019). URL: https://arxiv.org/pdf/1807.01069.pdf.

[8] Tsui-Wei Weng et al. "Evaluating the Robustness of Neural Networks: an Extreme Value Theory Approach". In: (2018). URL: https://openreview.net/pdf?id=BkUHlMZ0b.

[9] A. Rogier T. Donders et al. "Review: A gentle introduction to imputation of missing values". In: (2006).

[10] Julia Angwin et al. *Machine Bias*. 2016. URL: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[11] Matthew Arnold et al. "FactSheets: Increasing Trust in AI Services through Supplier's Declarations of Conformity". In: (2019). arXiv: 1808.07261 [cs.CY].

[12] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (2020), pp. 82–115.

[13] Yavar Bathaee. "The Artificial Intelligence Black Box And The Failure Of Intent and Causation". In: (2018).

[14] Reuben Binns. "On the Apparent Conflict Between Individual and Group Fairness". In: (2019).

[15] Philippe Bracke et al. "Machine learning explainability in finance: an application to default risk analysis". In: (2019).

[16] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[17] Stephen Buranyi. “Rise of the racist robots – how AI is learning all our worst impulses”. In: (2017).
[18] Niklas Bussmann et al. “Explainable machine learning in credit risk management”. In: *Computational Economics* 57.1 (2021), pp. 203–216.
[19] Nitesh Chawla. “C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure”. In: (2003).
[20] Alexis Cook and Var Shankar. *Synthetic Credit Card Approval*. 2021. URL: https://www.kaggle.com/alexisbcook/synthetic-credit-card-approval/activity.
[21] Sanjeeb Dash, Oktay Günlük, and Dennis Wei. “Boolean decision rules via column generation”. In: *arXiv preprint arXiv:1805.09901* (2018).
[22] Amit Datta, Michael Carl Tschantz, and Anupam Datta. “Automated Experiments on Ad Privacy Settings”. In: (2015).
[23] U.S. Department of Defense. *Defense Advanced Research Projects Agency*. 2021. URL: https://www.darpa.mil/program/explainable-artificial-intelligence.
[24] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. “Explainable artificial intelligence: A survey”. In: *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE. 2018, pp. 0210–0215.
[25] Cynthia Dwork. “Fairness Through Awareness”. In: (2011).
[26] EU. “Regulation (EU) 2016/679—general data protection regulation (GDPR). Official Journal of the European Union.” In: (2016).
[27] Fairlearn. *Improve fairness of AI systems*. 2021. URL: https://github.com/fairlearn/fairlearn.
[28] Stefan Feuerriegel, Mateusz Dolata, and Gerhard Schwabe. “Fair AI”. In: *Bus. Inf. Syst. Eng.* 62.4 (2020), pp. 379–384. DOI: 10.1007/s12599-020-00650-3. URL: https://doi.org/10.1007/s12599-020-00650-3.
[29] New York State Department of Financial Services. “Report on Apple Card Investigation”. In: (2021).
[30] Alex A Freitas. “Comprehensible classification models: a position paper”. In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10.
[31] Benyamin Ghojogh and Mark Crowley. *The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial*. 2019.
[32] google. *tensorflow*. 2021. URL: https://www.tensorflow.org/.
[33] Thomas Gramespacher and Jan-Alexander Posth. “Employing explainable AI to optimize the return target function of a loan portfolio”. In: *Frontiers in Artificial Intelligence* 4 (2021).
[34] David Gunning et al. “XAI—Explainable artificial intelligence”. In: *Science Robotics* 4.37 (2019).
[35] Xinjian Guo et al. “On the Class Imbalance Problem”. In: (2008).
[36] Marti A. Hearst et al. “Support vector machines”. In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.
[37] Michael Hind et al. “Experiences with Improving the Transparency of AI Models and Services”. In: (2019). arXiv: 1911.08293 [cs.CY].
[38] Michael Hind et al. “Experiences with improving the transparency of ai models and services”. In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–8.

[39] Robert R Hoffman et al. "Metrics for explainable AI: Challenges and prospects". In: *arXiv preprint arXiv:1812.04608* (2018).

[40] U.S. Department of Housing and Urban Development. *Housing Discrimination under the Fair Housing Act*. 2021. URL: https://www.hud.gov/program_offices/fair_housing_equal_opp/fair_housing_act_overview.

[41] IBM. *AI Explainability 360*. 2021. URL: https://aix360.mybluemix.net/?_ga=2.88374680.2057431906.1637135405-1405718511.1620651272.

[42] IBM. *AI Fairness 360*. 2021. URL: https://github.com/Trusted-AI/AIF360.

[43] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background". In: *International Journal of Engineering Research and Applications* 3.5 (2013), pp. 605–610.

[44] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[45] Todd Woodward Jennifer Whitman. "Self-selection bias in hypothesis comparison". In: (2010).

[46] Melike Demirici Joel Leupp and Jan Bauer. *Trusted-AI Git Repository*. 2021. URL: https://github.com/JoelLeupp/Trusted-AI.

[47] David G Kleinbaum et al. *Logistic regression*. Springer, 2002.

[48] Jon Kleinberg. "Inherent Trade-Offs in the Fair Determination of Risk Scores". In: (2016).

[49] Isabel Kloumann and Jonathan Tannen. *How we're using Fairness Flow to help build AI that works better for everyone*. 2021. URL: https://ai.facebook.com/blog/how-were-using-fairness-flow-to-help-build-ai-that-works-better-for-everyone.

[50] Nikita Kozodoi. "Fairness in Credit Scoring Assessment Implementations and Profit Implications". In: (2021).

[51] Karin Martin and Jack Glaser. "Racial profiling is a discriminatory practice that undermines fundamental civil rights while failing to promote law enforcement goals". In: 2012.

[52] Serguei Mokhov, Joey Paquet, and Mourad Debbabi. "The Use of NLP Techniques in Static Code Analysis to Detect Weaknesses and Vulnerabilities". In: (2014).

[53] Seyed-Mohsen Moosavi-Dezfooli. "DeepFool: a simple and accurate method to fool deep neural networks". In: (2016). URL: https://arxiv.org/pdf/1511.04599.pdf.

[54] Paul Mozur. "One Month, 500,000 Face Scans: How China Is Using A.I. to Profile a Minority". In: (2019). URL: https://www.nytimes.com/2019/04/14/technology/china-surveillance-artificial-intelligence-racial-profiling.html.

[55] Anthony J Myles et al. "An introduction to decision tree modeling". In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 18.6 (2004), pp. 275–285.

[56] David Wagner Nicholas Carlini. "Towards Evaluating the Robustness of Neural Networks". In: (2017). URL: https://arxiv.org/pdf/1608.04644.pdf.

[57] James R Norris. *Markov chains*. 2. Cambridge university press, 1998.

[58] Cathy O'Neil. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. 2016.

[59] Ziad Obermeyer et al. "Dissecting racial bias in an algorithm used to manage the health of populations". In: *Science* 366 (Oct. 2019), pp. 447–453. DOI: 10.1126/science.aax2342.

[60] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
[61] Plotly. *Plotly Dash*. 2021. URL: https://dash.plotly.com/.
[62] Executive Office of the President. “Big Data: A Report on Algorithmic Systems, Opportunity, and Civil Rights”. In: (2016).
[63] The Sentencing Project. “Racial Disparity in the Criminal Justice System”. In: (2008).
[64] pwc. *PWC Explainable AI*. 2017. URL: https://www.pwc.co.uk/audit-assurance/assets/explainable-ai.pdf.
[65] Jibran Rasheed Khan et al. “PREDICTIVE POLICING: A Machine Learning Approach to Predict and Control Crimes in Metropolitan Cities”. In: (Jan. 2019).
[66] John Richards et al. “A Methodology for Creating AI FactSheets”. In: (2020).
[67] Cynthia Rudin. “The age of secrecy and unfairness in recidivism prediction”. In: (2019).
[68] Mirka Saarela and Susanne Jauhiainen. “Comparison of feature importance measures as explanations for classification models”. In: *SN Applied Sciences* 3.2 (2021), pp. 1–12.
[69] Gustavo Saposnik. “Cognitive biases associated with medical decisions : a systematic review”. In: (2016).
[70] Iqbal Sarker, Hasan Furhad, and Raza Nowrozy. “AI-Driven Cybersecurity: An Overview, Security Intelligence Modeling and Research Directions”. In: (2021).
[71] Makoto Sato and Hiroshi Tsukimoto. “Rule extraction from neural networks via decision tree induction”. In: *IJCNN’01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*. Vol. 3. IEEE. 2001, pp. 1870–1875.
[72] P. Sattigeri et al. “Fairness GAN: Generating datasets with fairness properties using a generative adversarial network”. In: *IBM Journal of Research and Development* 63.4/5 (2019), 3:1–3:9. DOI: 10.1147/JRD.2019.2945519.
[73] *SGDClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.
[74] David Silver. “Mastering the Game of Go without Human Knowledge”. In: (2017).
[75] slundberg. *Shapley Additive Explanations*. 2021. URL: https://github.com/slundberg/shap/blob/master/docs/index.rst.
[76] Genevieve Smith. “What does fairness mean for machine learning systems”. In: (2019).
[77] T.Jayalakshmi. “Statistical Normalization and Back Propagation for Classification”. In: (2009). URL: http://www.ijcte.org/papers/288-L052.pdf.
[78] *Tesla’s Full Self-Driving tech keeps getting fooled by the moon, billboards, and Burger King signs*. URL: https://www.businessinsider.com/tesla-fsd-full-self-driving-traffic-light-fooled-moon-video-2021-7?r=US&IR=T.
[79] Jayne F Tierney and Lesley A Stewart. “Investigating patient exclusion bias in meta-analysis”. In: (2004).
[80] *Trusted AI Algorithm Files*. 2021. URL: https://github.com/JoelLeupp/Trusted-AI/tree/main/webapp/algorithms.
[81] *Trusted AI Algorithm Files*. 2021. URL: https://github.com/JoelLeupp/Trusted-AI/blob/main/webapp/configs/mappings/default.json.
[82] Muhammad Uzair. “Who Is Liable When a Driverless Car Crashes?” In: (2021).

[83] Sun-Chong Wang. "Artificial neural network". In: *Interdisciplinary computing in java programming*. Springer, 2003, pp. 81–100.

[84] Yang Xin et al. "Machine Learning and Deep Learning Methods for Cybersecurity". In: (2018).

[85] R. Caruana Y. Lou and J. Gehrke. "Knowledge Discovery and Data Mining". In: *ACM SIGKDD International Conference* (2012), pp. 150–158.

[86] Muhammad Bilal Zafar et al. "Fairness Beyond Disparate Treatment and Disparate Impact: Learning Classification without Disparate Mistreatment". In: 2017.

[87] Richard Zemel et al. "Learning fair representations". In: *30th International Conference on Machine Learning, ICML 2013* (Jan. 2013), pp. 1362–1370.

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

ART Adversarial Robustness Toolbox.

CLEVER Cross Lipschitz Extreme Value for Network Robustness.

CNN Convolutional Neural Network.

DI Disparate Impact.

DL Deep Learning.

DoF Degrees of Freedom.

DP Demographic Parity.

DT Decision Tree.

ER Empirical Robustness.

FPR False Positive Rate.

GAN Generative Adversarial Network.

GDPR General Data Protection Regulation.

IoT Internet of Things.

KNN K-Nearest Neighbor.

LR Logistic Regression.

ML Machine Learning.

NN Neural Network.

RFC Random Forest Classifier.

SGD Stochastic Gradient Descent.

SPD Statistical Parity Difference.

SVC Support Vector Classifier.

SVM Support Vector Machines.

TPR True Positive Rate.

XAI Explainable Artificial Intelligence.

Glossary

Artificial Intelligence: is a discipline concerned with building computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

Machine Learning: refers to the use and development of systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data.

Deep Learning: is a subfield of machine learning concerned with algorithms and models inspired by the function of the human brain called artificial neural networks. These are built up in layers and when the number of layers becomes large, this is called Deep Learning.

List of Figures

List of Tables

List of Algorithms

Listings

Appendix A

Example - Trusted AI Report

TRUSTED AI

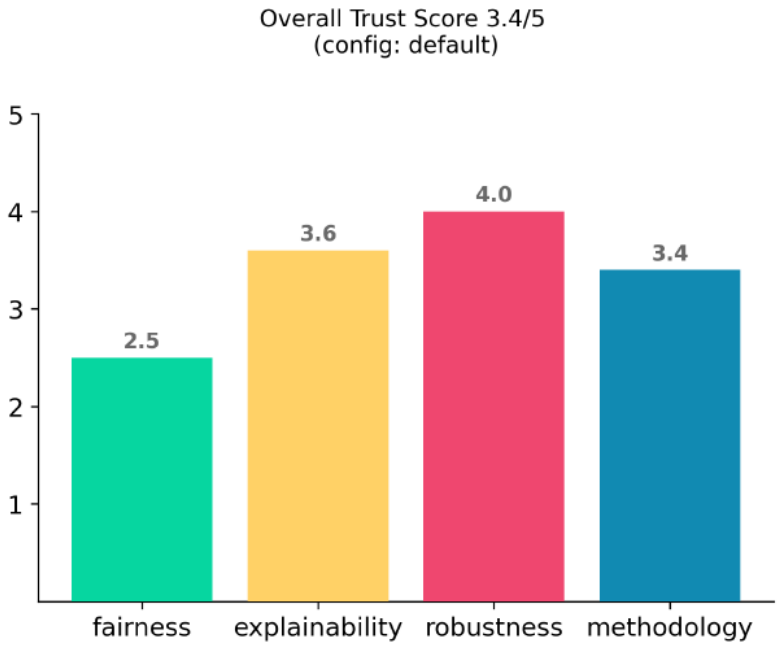
Report

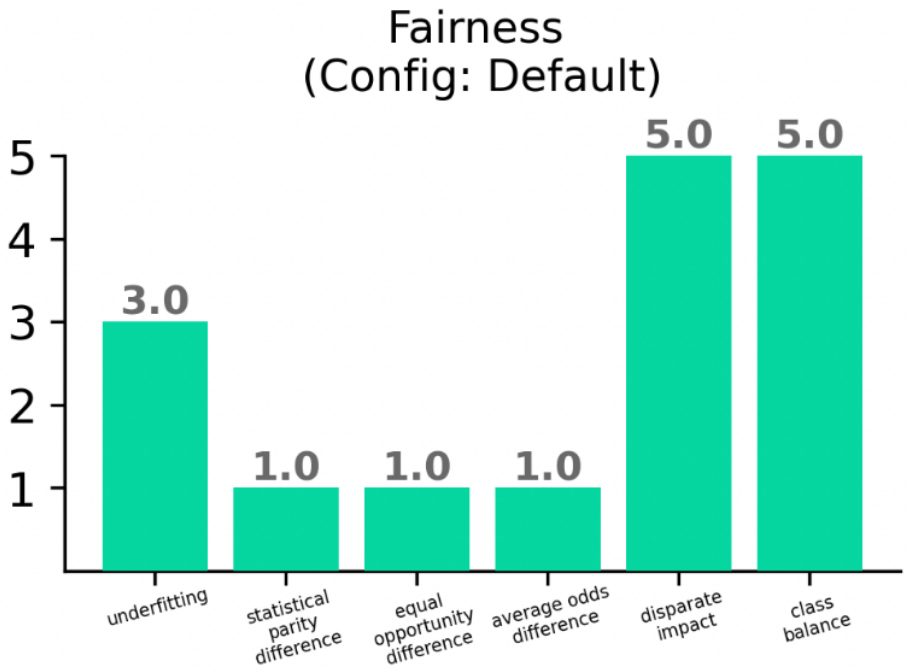
Model Information

Model Name:	Jan's Random Forest Classifier	Purpose Description:	Attack Classification
Domain Description:	IT Security	Training Data Description:	Alberto's classification data
Model Information:	Sklearn Random Forest Classifier for IT Security Incident Classification scenario	Target Column:	label
Authors:	Jan Bauer	Contact Information:	jan.bauer@uzh.ch

Performance Of The Model

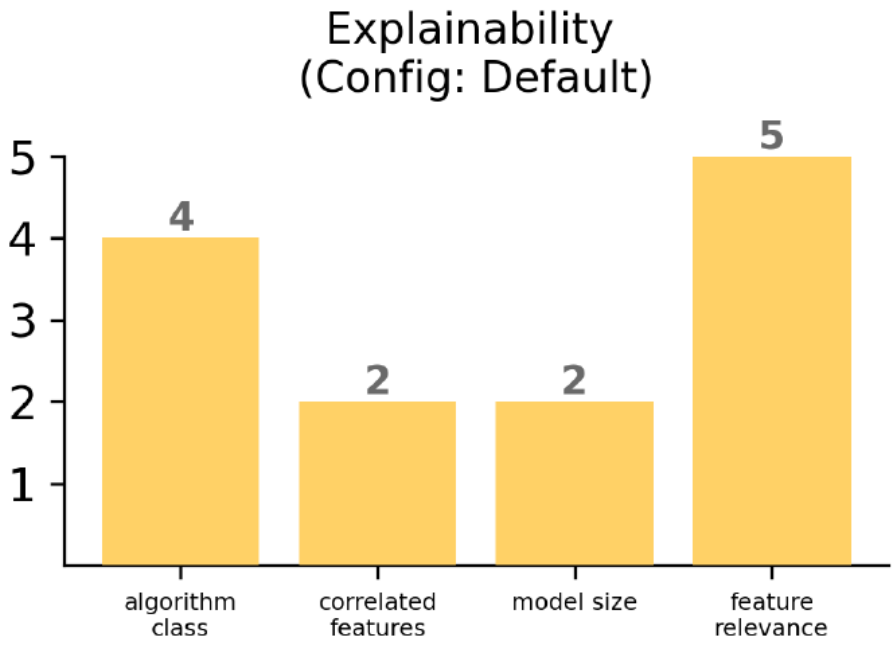
Accuracy:	0.88		
Global Recall:	0.88	Class Weighted Recall:	0.88
Global Precision:	0.88	Class Weighted Precision:	0.88
Global F1 Score:	0.88	Class Weighted F1 Score:	0.87





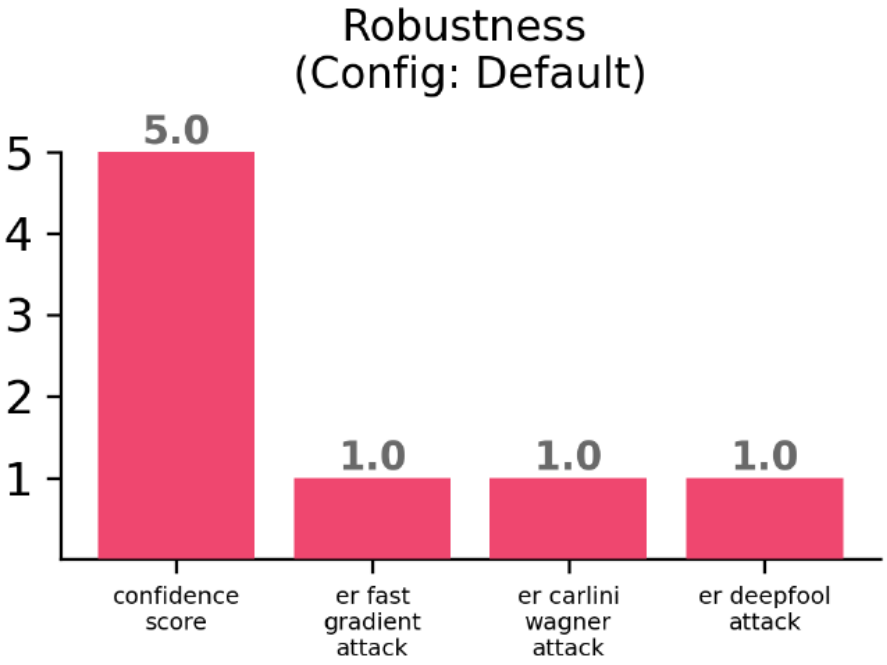
Fairness Properties

Test Accuracy:	88.06%	Statistical Parity Difference Formula:	Statistical Parity Difference = |Favored Protected Group Ratio - Favored Unprotected Group Ratio|
Statistical Parity Difference:	46.20%	Equal Opportunity Difference Formula:	Equal Opportunity Difference = |TPR Protected Group - TPR Unprotected Group|
Equal Opportunity Difference:	45.45%	Average Odds Difference Formula:	Average Odds Difference = |0.5*(TPR Protected - TPR Unprotected) + 0.5*(FPR Protected - FPR Unprotected)|
Average Odds Difference:	22.36%	Disparate Impact Formula:	Disparate Impact = Protected Favored Ratio / Unprotected Favored Ratio
Disparate Impact:	10.55		



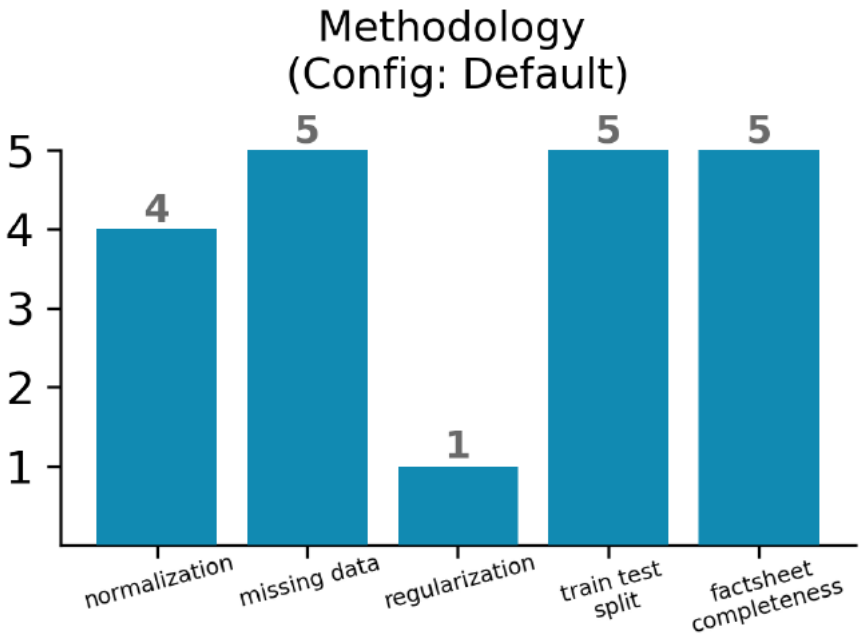
Explainability Properties

Model Type:	RandomForestClassifier	Percentage Of Highly Correlated Features:	39.73%
Number Of Features:	73	Number Of Outliers In The Importance Distribution:	0
Percentage Of Feature That Make Up Over 60% Of All Features Importance:	24.66%		



Robustness Properties

Average Confidence Score:	86.33%	Fgm Before Attack Accuracy:	92.00%
Fgm After Attack Accuracy:	12.00%	Fgm Proportional Difference (After-Att Acc - Before-Att Acc)/Before-Att Acc:	86.96%
Cw Before Attack Accuracy:	100.00%	Cw After Attack Accuracy:	0.00%
Cw Proportional Difference (After-Att Acc - Before-Att Acc)/Before-Att Acc:	100.00%	Df Before Attack Accuracy:	100.00%
Df After Attack Accuracy:	0.00%	Df Proportional Difference (After-Att Acc - Before-Att Acc)/Before-Att Acc:	100.00%



Methodology Properties

Mean Of The Training Data:	-0.00	Standard Deviation Of The Training Data:	1.00
Mean Of The Test Data:	-0.03	Standard Deviation Of The Test Data:	0.88
Normalization:	Training data are standardized	Number Of The Null Values:	0
Regularization Technique:	none	Train Test Split:	80.00/20.00
Factsheet Property Model Name:	present	Factsheet Property Purpose Description:	present
Factsheet Property Domain Description:	present	Factsheet Property Training Data Description:	present
Factsheet Property Model Information:	present	Factsheet Property Target Column:	present
Factsheet Property Authors:	present	Factsheet Property Contact Information:	present