

TDK-dolgozat

András Gergely Deé-Lukács

2024.



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Artificial Intelligence and Systems Engineering

System-level impact analysis of embedded AI components

Scientific Students' Association Report

Author:

András Gergely Deé-Lukács
IV. évfolyam

Advisor:

Prof. András Pataricza,
egyetemi tanár

András Földvári,
doktorandusz

2024.

Contents

Kivonat	i
Abstract	ii
1 Introduction	1
1.1 Fundamental concepts of dependability	1
1.2 Embedded AI models in CPS	2
1.3 Verification and Validation	3
1.4 Main objective	3
2 Qualitative modelling	5
2.1 Introduction to qualitative clustering	6
2.2 Methods of clustering	6
2.3 Qualitative transformations	7
2.3.1 Clustering in qualitative modelling	7
2.3.2 Clustering continuous values	7
2.3.3 The term qualitative landmark	8
3 Machine learning	9
3.1 Supervised learning	9
3.2 Confusion matrices in ML	10
3.3 Risk-driven ML	11
4 Explainable AI	13
4.1 Explanatory Model Analysis	14
4.2 Typical xAI models	14
4.2.1 Decision Tree	15
4.2.2 Boolean Rules via Column Generation	15
4.2.3 Repeated Incremental Pruning to Produce Error Reduction	16
4.2.4 Interpretable Model Differencing	17

4.3	Qualitative modelling in xAI	19
4.4	Ruleset to qualitative model	19
5	Impact Analysis of ML models	21
5.1	Challenges	21
5.1.1	Different impact of FP and FN predictions	22
5.1.2	Highly imbalanced dataset	22
5.2	Error Propagation Analysis	22
5.3	Context-independent model analysis	23
5.3.1	Dataset-side improvement	24
5.3.2	Preprocessing imbalanced datasets	24
5.3.3	Model-side improvement	25
5.4	Context-dependent model analysis	25
5.4.1	Faults projected forward	27
5.4.2	Difference modeling	28
5.4.3	xAI Surrogate model	30
5.4.4	Faults projected backwards	32
6	Conclusion and Future work	35
	Bibliography	35
	Appendix	40
A.1	Abbreviations	40
A.2	Syntax of the rulesets	41
A.3	Experiments using the Adult Dataset	42
A.3.1	Adding semantic redundancy	43
A.3.2	Explainable difference model	43
A.3.2.1	Training the DT model	44
A.3.2.2	Training the BRCG	44
A.3.2.3	Creating the difference model	45
A.4	Experiments using the APS Failure at Scania Trucks dataset	47
A.4.1	High level of imbalance	47
A.4.2	Difference model on the positive samples	47
A.5	Experiments using the MetroPT-3 dataset	48
A.5.1	Training an advanced model with preprocessed data	49
A.5.1.1	Preprocessing	49
A.5.1.2	Training a neural network with preprocessed data	50

A.5.2	Difference model on positive samples	50
A.5.2.1	IMD difference model	51
A.5.2.2	XOR difference model	51
A.5.2.3	From DT difference model to qualitative model	52
A.5.2.4	Difference model with 8 labels	55
A.5.3	Surrogate model creation	56
A.5.4	Evaluation of the surrogate model	57
A.5.5	Critical samples and model improvement	57

Kivonat

A dolgozat célja a fizikai folyamatok beágyazott ML-modellvezérelt vezérlőiből eredő veszélyek kezelése. A kiber-fizikai rendszerek (CPS) bizonyos operatív technológiákat (OT) információtechnológiával (IT) vezérelnek, jellemzően felügyeleti vezérlést valósítva meg. A modern CPS-architektúrák gyakran ML modelleket ágyaznak be, hogy csökkentsék a gyakran igen összetett vezérlőszoftverek szükségességét. A beágyazott ML-modellek ellenőrzése és validálása azonban kihívást jelentő és részben megoldatlan probléma. A kulcsprobléma a beágyazott ML-modell hibáinak az OT-re gyakorolt esetleges veszélyes hatásainak meghatározása. A cél egy univerzális és magyarázható megoldás kidolgozása, ami megmutatja az ML model gyengeségeit. Új hatáselemző algoritmusom a magyarázható mesterséges intelligencia (xAI) technikákat és a kvalitatív logikai következtetést integrálja egy hibaterjedést elemző (EPA) algoritmusba, amely meghatározza a potenciális rendszerszintű veszélyeket, akár már tervezési időben is. Ez segít jobban felkészülni az esetleges veszélyhelyzetekre, így csökkentve a kockázatokat, ezzel biztosítva a szolgáltatásbiztonságot.

Abstract

This thesis aims to manage hazards originating in embedded ML-model-driven controllers of physical processes. Cyber-physical systems (CPS) control some operational technology (OT) by information technology (IT), typically implementing supervisory control. Modern CPS architectures frequently embed models generated by machine learning processes to reduce the need for frequently highly complex control software. However, verifying and validating embedded ML models are challenging and partly unsolved problems. The focal problem is the estimation of the hazardous impacts of errors of the embedded ML model on the OT. The goal is to develop a universal and explainable solution that identifies the weaknesses of the ML model. My novel impact-analysis algorithm integrates explainable AI (xAI) techniques with qualitative logic reasoning into an error propagation analysis (EPA) algorithm estimating the potential system-level hazards. This method helps mitigate risks by preparing for potentially hazardous situations, thus ensuring dependability.

Chapter 1

Introduction

Cyber-physical systems (CPS) are integrated networks of physical components and computational elements that interact closely to monitor, manage, and control physical processes. Their main goal is to control some operational technology (OT) by information technology (IT), typically implementing supervisory control. These systems integrate both digital and physical components, creating a dynamic ecosystem that enables real-time interaction, data collection, and analysis to guide decision-making.

One of the defining features of CPS is their ability to operate autonomously or semi-autonomously, leveraging adaptive algorithms and predictive models. These capabilities allow CPS to anticipate issues before they arise and to adjust system parameters to optimize performance. Additionally, CPSs rely on networked communication to ensure the exchange of data and instructions between different components, providing unified monitoring and control.

CPSs have a wide range of applications. They are used in industrial control systems, from smart factories, where they enhance production efficiency and product quality, to autonomous vehicles that navigate complex environments, healthcare devices that monitor patient conditions in real time, and energy grids that adapt to fluctuating demand. These systems represent the forefront of technological advancement in fields that demand precision, reliability, and adaptability.

CPSs increasingly use embedded AI components, whose failures can have serious consequences due to the interconnection of IT and OT. It is therefore necessary to apply solutions that allow the safe usage of embedded AI/ML components in critical CPS systems. This report presents the application of explainable approximate AI models for an exhaustive assessment of the faults in critical cyber-physical systems built around embedded AI.

1.1 Fundamental concepts of dependability

Dependability [2] represents a critical system property defined by the ability of a system to deliver service that can be justifiably trusted by users, extending far beyond traditional performance metrics. It integrates multiple essential attributes including *reliability*, *availability*, *safety*, *security*, *survivability*, and *maintainability*. The concept consists of three parts: the *threats* to, *attributes* of, and the *means* by which dependability is attained.

The threats manifest through a cascading progression: *faults*, *errors*, and ultimately *failures*. A fault is the hypothesized or identified cause of an error, representing a defect or

abnormality within the system’s structure or functionality. When a fault becomes active (usually at a component level), it generates an error, which is a specific state within the system that has the potential to trigger a subsequent failure. A failure occurs when the system deviates from its specified behavior.

In engineering terms, *hazard* is any potential source of harm or adverse effect within a system, typically a condition or event that, if not controlled, could lead to a failure or an accident. A *risk* is the combination of the likelihood of a hazardous event occurring and the severity of its potential consequences as defined by different standards.

Dependability is measured through multiple interconnected attributes. *Availability* quantifies the system’s readiness to provide service when required. *Reliability* represents the probability of continuous correct operation over a specified time interval. *Safety* ensures the system does not cause catastrophic damage or harm. *Confidentiality* protects sensitive information from unauthorized access, while *integrity* guarantees the accuracy and consistency of data. *Maintainability* describes the system’s capacity for effective maintenance and repair.

To achieve and enhance dependability, system designers employ four fundamental means: *fault prevention*, *fault tolerance*, *fault removal*, and *fault forecasting*. Fault prevention focuses on trying to eliminate the occurrence or introduction of faults during the design and implementation phases. Fault tolerance enables the system to continue operating correctly despite the presence of faults, often through redundant mechanisms or self-healing capabilities. Fault removal focuses on trying to reduce the number or severity of faults. Fault forecasting uses predictive techniques and probabilistic models to estimate the present number or anticipate future incidences and their likely consequences.

Ultimately, dependability transcends mere technical performance, representing a continuous process of delivering reliable, safe, and trustworthy technological solutions that can be consistently and confidently utilized across diverse and demanding environments.

In practice, the method for ensuring the dependability of CPS consists of putting *requirements* on its service interfaces that must be met. These requirements are 1) *functional*, describing the system’s primary goals, and the reason for the system’s existence and 2) *extra functional*, expectations regarding the system’s quality, such as safety, security, or reliability requirements.

1.2 Embedded AI models in CPS

Embedded AI models within CPS play a crucial role in combining computational and physical processes to form adaptive frameworks where they enhance several aspects like security and real-time responsiveness.

By integrating sophisticated algorithms and machine learning models directly into the physical system infrastructure, CPS can be leveraged across various sectors, such as health-care, transportation, and industrial manufacturing, to drive advanced operational capabilities.

Embedded models enable features like predictive maintenance, anomaly detection, and adaptive decision-making, which allows CPSs to anticipate and respond to real-world events with minimal human intervention. For example, predictive maintenance in industrial environments uses embedded analytics to detect equipment degradation or failure early, increasing availability, making it easier to maintain, and optimizing productivity. Similarly, anomaly detection algorithms continuously monitor system behavior to iden-

tify irregularities, enabling timely responses that prevent potential failures or security breaches. However, to ensure the dependability of these embedded models, robust verification and validation methods are essential. This ensures, the model performs well within its context and complies with system-level requirements as much as possible.

In threat detection and cybersecurity, embedded models perform behavior analysis, anomaly detection, or intrusion detection tasks to safeguard CPS from unauthorized access or malicious activity.

In real-time applications, AI-driven traffic management systems, for instance, optimize traffic flow by analyzing real-time data to reduce congestion and improve road safety. Autonomous vehicles utilize ML to interpret their surroundings and make informed decisions. In industrial contexts, methods like predictive maintenance, or big data analytics are used to continuously refine production processes.

1.3 Verification and Validation

Verification and *validation* are complementary processes in systems engineering that ensure the quality, correctness, and appropriateness of a product throughout its development lifecycle. [7] Verification focuses on the critical question, "*Are we building the product right?*" It is a systematic evaluation to confirm that the system fully complies with its predefined specifications and design requirements. This process can be conducted through both static methods, such as reviewing the code or documentation, and dynamic methods involving active testing and execution.

In contrast, validation addresses the fundamental question, "*Are we building the right product?*" This process goes beyond mere technical compliance, concentrating on determining whether the system genuinely meets the end-user's actual needs, expectations, and intended purpose. Validation ensures that the product solves the right problem and delivers meaningful value to its intended users.

The techniques employed in verification and validation are comprehensive and multifaceted, ranging from detailed requirements reviews and design assessments to rigorous testing methodologies. These include a series of increasingly comprehensive evaluations, including requirements review, design review, code review, unit testing, integration testing, system testing, and acceptance testing. For critical systems with high-stakes implications, formal methods may be employed to mathematically prove system correctness. By meticulously examining both the technical implementation and the ultimate user value, verification and validation provide a robust framework for ensuring system quality and dependability.

1.4 Main objective

The assessment of a mission-critical CPS has to be as exhaustive as possible as the evaluation of risks involves working with hazards of low probabilities, but of a high impact. For instance, in a safety-critical application, the probability of a hazardous situation should be around 10^{-9} and the probability of a disaster around 10^{-12} . This necessitates the system to undergo mathematical correctness proofs including potentially faulty cases as well. This also implies that the verification can only practically occur at an abstract level, where the abstraction is preferably discrete, as formal modeling tools are available for

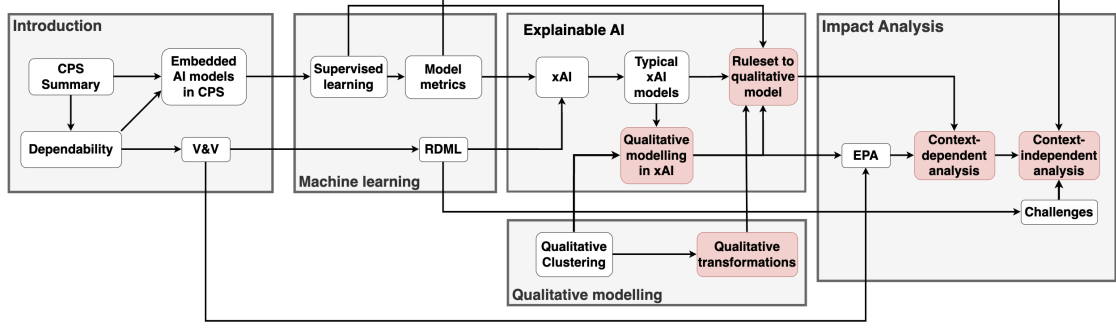


Figure 1.1: Report structure

discrete models. In general, in the case of complicated problems, it is not feasible to prove correctness analytically within a continuous domain.

Our goal is to estimate the system-wide impacts of local faults in embedded AI components in an exhaustive way to assess and guarantee the conformance of the entire system to the functional and extra-functional requirements by formal methods. I would like to also provide a *universal* and *explainable* method for design time impact analysis of errors especially training errors in the AI/ML components, to anticipate, prepare for, and mitigate these faults early in the development process. This is achieved by integrating xAI into EPA.

Concerning the difficulties of modeling complex IT infrastructure, these processes apply qualitative abstraction. This method does not lose critical information while allowing the application of efficient, discrete formal methods for analysis. It also addresses the problem of imbalanced datasets, by approximating the underrepresented samples. The basic idea is the comparison of refined models used in production with xAI interpretable-by-design ones, which support qualitative abstraction.

The methodological pillars of our methods are the following:

- Approximation and evaluation of the embedded model by xAI techniques
- Qualitative abstraction and reasoning offer models that can be evaluated using exhaustive methods and are also easy to map xAI models to them.
- Based on a known EPA method relying on formal reasoning, estimation of an upper approximation in the qualitative domain of the cases potentially violating requirements at the IT/OT interface.

The structure of the report and the dependencies of the topics are shown in Figure 1.1, where the red nodes represent my contribution. **My contribution** manifests in the following: 1) **I created an interface between xAI models and EPA**. This means implementing the conversion of rulesets to qualitative models using interval logic. 2) I addressed the challenges of imbalanced datasets by trying out undersampling, oversampling and **invented a custom boosting technique (A.5.3)**. 3) To address the problem of different impact of misclassifications **I created approximate models, I compared them to the advanced models and I determined the undetected critical scenarios using the workflow described in 5.4.4**. 4) **I created a model to approximate these scenarios, to use it for improvement in future work**.

Chapter 2

Qualitative modelling

Qualitative Modeling [16] (QM) is an approach to representing and reasoning about continuous aspects of systems, such as quantities, motion, space, and time in a symbolic, human-like way. Unlike traditional mathematical models, such as differential equations requiring precise numerical values, QM thrives on abstracting information and operating with minimal details, letting people intuitively understand changes in quantities without specific numeric knowledge. This approach is often sufficient for practical decision-making; for instance, recognizing the risk of a fragile item falling off a table motivates preventive action. However, in technical fields, qualitative insights can guide further, more detailed analyses.

The core goal of qualitative modeling is to replicate human-like reasoning capabilities for understanding continuous systems, which have significant applications in areas such as science, engineering, education, and decision support. The approach emphasizes both the representation of information and reasoning methods, where both aspects are integral and interdependent.

There are three key principles that define qualitative modeling:

- **Discretization:** Dividing continuous properties into meaningful categories or qualitative values. This simplifies representation and enables reasoning with limited information. For example, instead of representing temperature as a continuous range, QM might categorize it as COLD, WARM, or HOT based on relevant thresholds like freezing and boiling points.
- **Relevance:** Selecting appropriate discretizations to the task at hand. This ensures that the qualitative model captures the essential aspects of the system relevant to the reasoning process. For instance, when assessing drinkability, finer temperature subdivisions within the WARM category might become necessary.
- **Ambiguity:** Acknowledging the inherent uncertainty arising from working with high levels of abstractions. QM often yields multiple possible outcomes, reflecting the lack of complete information. This ambiguity makes QM a valuable tool for problem framing and identifying potential behaviors, which can then be refined using more detailed models.

The fundamental steps in basic qualitative reasoning are the following:

- **Model formulation:** Constructing a model using an input description known as a *scenario*. Information about potential entities and phenomena is stored as *model*

fragments within a *domain theory* library. The model for the scenario is then assembled from these fragments through a reasoning process called *model formulation*, which uses both the scenario details and task-specific constraints.

- **Elaborating a qualitative state:** Combining active model fragments with qualitative values for all of the system parameters defines a *qualitative state*. This represents a specific state of the system, acknowledging potential incompleteness in the initial description
- **Qualitative simulation:** Analyzing state transitions to predict possible future behaviors. This involves identifying potential next states and understanding the causal relationships that drive the system's evolution. *Envisioning*, a key aspect of simulation, explores all possible behavioral categories from a given initial state.

2.1 Introduction to qualitative clustering

Qualitative clustering involves organizing and analyzing qualitative data by considering entities or patterns with similar behavior equivalent, thus grouping them together. This process enhances understanding and interpretation by categorizing data based on shared characteristics, allowing researchers to identify underlying patterns and relationships within unlabelled datasets. Through clustering, hidden structures within the data can emerge, offering insights that may otherwise remain unnoticed. For instance, qualitative clustering can reveal behavior patterns directly through reasoning with qualitative models, capturing key dynamics within complex systems without requiring precise numerical inputs.

Qualitative clustering has diverse applications across various fields, where efficient fault reporting, real-time monitoring, improved communication, and informed decision-making are essential. By grouping similar occurrences or behaviors, clustering can streamline the identification of issues in systems and environments, enabling quick responses. In decision support, qualitative clusters aid by summarizing and simplifying complex data, making it easier for stakeholders to understand trends and make informed choices. In an industrial example [35], qualitative clustering is applied by grouping similar failure patterns and observable component states (like "object visible," "conveyor stopped," or "robot arm moved") to streamline fault diagnosis. This clustering enables the system to identify recurring fault scenarios without requiring precise numerical data, improving the efficiency and interpretability of diagnostic rule generation.

2.2 Methods of clustering

Clustering methods are diverse, each with strengths and weaknesses, making them suitable for specific data and analytical goals. They can be categorized in the following approaches:

- **Hierarchical clustering:** Creates a tree-like structure of clusters. The *agglomerative* approach (bottom-up) starts with individual data points as clusters and merges them based on a linkage criterion until a stopping criterion is met. An example is the AGNES [26] (agglomerative nesting) algorithm. The *divisive* approach starts with all data points in one cluster and recursively splits them. It is computationally intensive but offers detailed insights. Examples of hierarchical clustering are concept lattices in Formal Concept Analysis [4].

- **Partitional clustering:** Involves dividing the dataset into distinct clusters, where each data point belongs to exactly one cluster. For instance, the k-means algorithm [25] partitions the data into K clusters by minimizing the variance within each cluster.
- **Density-based clustering:** Identifies clusters based on the density of data points in a region. Useful for discovering clusters with arbitrary shapes. For instance, the DBSCAN [14] algorithm groups densely packed points and labels outliers as noise. It relies on parameters like epsilon (neighborhood radius) and the minimum number of points per cluster.
- **Model-based clustering** [31]: Assumes the data arises from a mixture of underlying probability distributions. Uses algorithms like Expectation-Maximization (EM) [13] to estimate distribution parameters and identify clusters.
- **Specialized methods:** There are clustering methods tailored to a specific application, such as biological data analysis. For instance, biclustering [8] clusters both samples and variables simultaneously, suitable for gene expression data. Overlapping clustering [24] allows data points to belong to multiple clusters, which is useful for analyzing community structures in social networks.

2.3 Qualitative transformations

2.3.1 Clustering in qualitative modelling

Let $X = \{x\}$ be the domain of feature to be clustered and Q the *qualitative domain* of the clustered feature so that $Q = \{q\}, Q \subset \mathbb{Z}$. As a first step X_i, X_j non-empty subsets of X are selected, so that $X_i \cap X_j = \emptyset$ and $X_i, X_j \neq \emptyset$. These two subsets contain elements with *similar behavior*. This means these elements only differ in aspects, which are not as relevant. Relevance is defined by the question to be answered using the qualitative model.

Let $cf : X \mapsto Q$ be a *clustering function* that maps each subset X_i to a *qualitative value* q , so that $cf(X_i) = q$.

2.3.2 Clustering continuous values

Let $X \subseteq \mathbb{R}$ be the domain of a continuous feature and $I = I_1, I_2, \dots, I_n \subset \mathbb{R}$ real intervals. [28] These intervals represent *partitions* of the feature domain, so that:

$$\forall I_i, I_j : I_i \cap I_j = \emptyset; \bigcup_i I_i = X$$

These partitions are selected in a way that the values they contain represent a similar behavior in the system. Note that, this behavior is also defined by the task at hand.

There's an *interval membership function* that simply assigns each number to an interval.

$$mb : X \mapsto I : mb(x) = I_i \iff x \in I_i$$

In this case, the cf clustering function assigns a qualitative value to an element, the value corresponding to a one-on-one mapping with an interval partition of the feature domain.

$$x \in X : cf(x) \in Q \wedge cf(x_1) = cf(x_2) \iff \exists I_i : mb(x_1) = mb(x_2)$$

2.3.3 The term qualitative landmark

A *qualitative landmark* of two qualitative values refers to the boundary value between the two corresponding neighboring interval partitions. Let I_1, I_2 be two neighboring intervals so that:

$$I_1, I_2 \in \mathbb{IR} : I_1 = [a, l], I_2 = [l, b]$$

If q_1 and q_2 are two neighboring clusters and they are mapped to the intervals I_1 and I_2 i.e.:

$$x_1 \in I_1, x_2 \in I_2 : cf(x_1) = q_1, cf(x_2) = q_2,$$

then l is the qualitative landmark between q_1 and q_2 .

Chapter 3

Machine learning

3.1 Supervised learning

This research report mainly focuses on the application and analysis of *supervised learning* ML models in CPS. Supervised learning is a type of machine learning where an algorithm learns from a *labeled dataset* $D = \{(x, y)\}^n$ to make *predictions* or decisions. In supervised learning, each data point $x \in X_{data}$ is paired with a *label* or output $y \in Y_{data}$, which serves as the "supervision" or guidance that the model uses to learn patterns. The variables in a dataset are called *features*. A dataset contains *samples*. A sample is a data point represented by a d-dimensional vector containing the value of each feature so that $X_{data} \subset \mathbb{R}^d$, where X_{data} contains all the unlabeled samples and Y_{data} contains each label associated with the samples. Each sample represents an observation of the phenomenon based on which a decision has to be made. The goal is for the model to learn the relationship between the input data and the labeled outputs, allowing it to make accurate predictions on new, unseen data. This relationship is represented by the function $learn : X_{data} \times \theta \mapsto Y_{data}$ where θ denotes the model parameters.

The process of supervised learning consists of 4 steps:

1. **Training data:** The dataset D is built from the collected samples of the phenomenon.
2. **Learning process:** The function $learn$ is learned by examining the input-output pairs to find patterns or relationships within the data.
3. **Testing and Validation:** Once trained, the model is tested on new data (validation or test set) to see how well it generalizes
4. **Prediction:** After training and validation, the model can be used on new, unlabeled data to make predictions

Y_{pred} contains all of the predictions of the model for the samples from a given dataset D . If x is a sample and y is the prediction of the model for x , then: $y \in Y_{pred} \iff y = learn(x, \theta) \wedge x \in X_{data}$

Supervised learning mainly consists of two tasks:

- **Classification:** Predicting a category or class, i.e. $Y_{data} \subset \mathbb{Z}$, such as identifying whether a human gets \$50k a year or not, or detecting objects in an image.

- **Regression:** Predicting a continuous value, i.e. $Y_{data} \subset \mathbb{R}$, such as forecasting stock prices or estimating the fuel consumption of a car.

In supervised learning, the function $loss : Y_{pred} \times Y_{data} \mapsto \mathbb{R}$ is used to measure the difference between a predicted output of a model and an actual target value in the training data. It is called *loss function* and it is used when referring to the error for a single training example, it is also known as *training error*.

The *cost function* also known as risk-function is used to refer to an average of the loss function values over an entire data distribution, i.e. the expected loss over all possible values of x . This value is also known as *generalization error* because it shows how well the model is able to generalize over a set of data.

$$cost(\theta) = \mathbb{E}[loss(Y_{data}, Y_{pred})] = \iint loss(y, learn(x, \theta))p(x, y) dx dy$$

Where $p(x, y)$ is the joint probability distribution of X_{data} and Y_{data} .

In practice, this value is often calculated using the *empirical cost function*, since the true distribution $p(x, y)$ is often not known, thus a uniform distribution is assumed.

$$cost(\theta) = \frac{1}{n} \sum_{i=1}^n loss(y, learn(x, \theta))$$

Overfitting means that the model produces highly accurate results over the training set, but it is inaccurate in predicting new data. This means it has learned the training data too specifically and may struggle with new data. In other words, the training errors are very low, but the generalization error is high.

Underfitting means the exact opposite, when the model has not learned the training data adequately so it has a high training error.

3.2 Confusion matrices in ML

		Predicted label	
		Positive (PP)	Negative (PN)
True label	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Figure 3.1: Confusion Matrix

In the field of machine learning, a confusion matrix is a table that helps evaluate the performance of a classification model by showing the relationship between predicted and actual values. The structure of a confusion matrix is shown in Figure 3.1. It has the 4 following key components:

- *True Positive Prediction* (TP): Cases correctly predicted as positive
- *False Positive Prediction* (FP): Cases incorrectly predicted as positive i.e. Type I error
- *False Negative Prediction* (FN): Cases incorrectly predicted as negative i.e. Type II error
- *True Negative Prediction* (TN): Cases correctly predicted as negative

The positive samples are denoted by P, the negative samples with N, the predicted positives by PP, and the predicted negatives with PN. This way, the following equations hold: $P = TP + FN$, $N = FP + TN$, $PP = TP + FP$, $PN = FN + TN$

One of the most straightforward metrics of an ML model is *accuracy*. This measures the proportion of the correct predictions out of all predictions.

$$accuracy = \frac{TP + TN}{P + N}$$

The accuracy of positive predictions is called *precision*. It measures, how many of the samples predicted positive are actually positive.

$$precision = \frac{TP}{PP} = \frac{TP}{TP + FP}$$

The *sensitivity*, also known as recall, measures the ability of the model to find all positive cases. It measures from all the positive classes, how many we predicted correctly. The specificity measures the same rate, but for negative samples, i.e. from all the negative samples how many we predicted correctly.

$$sensitivity = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$specificity = \frac{TN}{N} = \frac{TN}{FP + TN}$$

The metric *F-score* is the weighted harmonic mean of precision and sensitivity, where the relative importance of the two metrics can be adjusted with a parameter β .

$$F_{\beta} = (1 + \beta^2) \cdot \frac{precision \cdot sensitivity}{(\beta^2 \cdot precision) + sensitivity}$$

A special case of F-score is the *F1-score*, when $\beta = 1$

Generally in a mathematical sense FN and FP predictions are equally penalized. However, in a technical system, the implications differ. For instance, a FP prediction decreases the availability of a system (false alarm), a FN prediction can lead to a hazard (escaping fault). Therefore, specialized techniques have been developed to account for these consequences.

3.3 Risk-driven ML

In ML, the average of the errors (misprediction) is named risk. This is accurate when all outcomes have equal probability and significance. However, in a technical system, this is not actually uniform for two reasons. First, the likelihood of a given misprediction

occurring or not occurring is not equal. Second, not all errors are identical, meaning that here, hazards and technical risks are considered in a weighted manner. In this sense, the evaluation of critical ML models embedded within CPS differs significantly. Therefore, in this report, the term *risk* always refers to technical risks in CPS.

Risk-driven machine learning (RDML) represents a critical approach to developing and deploying ML technologies that explicitly emphasizes comprehensive risk management throughout the entire machine learning lifecycle. Unlike traditional ML methodologies that primarily focus on model performance, this approach systematically integrates risk identification, assessment, and mitigation as core components of the development process.

The fundamental objective is to proactively address potential failures and unintended consequences before they can affect the dependability of the system. This is particularly crucial in domains such as failure prediction and anomaly detection, where the consequences of model errors can often be critical. A *rare event* in a safety-critical system is a low-probability but high-consequence occurrence that, despite its infrequent nature, can lead to catastrophic outcomes if it occurs. In RDML in CPS, there are two main difficulties to overcome. The first one is that these rare events have relatively few appearances in the training data, making it difficult for the models to learn them. The second one is that these rare events have a high level of impact; therefore, their technical risk is high. RDML examines potential risks across multiple dimensions, including technical performance, ethical implications, and regulatory compliance.

Key aspects of risk-driven ML include *risk identification*, where potential fault modes or vulnerabilities are comprehensively mapped out. *Risk assessment* involves quantifying the potential impact and likelihood of these identified risks, enabling prioritized mitigation strategies. *Continuous monitoring* ensures that risks are tracked and managed dynamically throughout the model’s operational lifecycle. *Ethical considerations* involve incorporating ethical principles and guidelines into the ML development process to ensure that the model respects societal values such as fairness or privacy. *Transparency and explainability* enables people to understand, trust, and validate the model’s decisions.

Chapter 4

Explainable AI

Machine learning models have become increasingly complex, often containing thousands of parameters that make their decision-making processes opaque. The need for model explainability is driven by several key factors, such as Legal requirements for the "right to explanation" [18], ethical considerations in AI decision-making, model debugging or improvement needs, building trust in AI systems, and enhancing model-human interaction. This has led to the emergence of *Explainable AI* (xAI) and *Explanatory Model Analysis* [6] as crucial fields in modern machine learning. The goal is to create AI systems whose decision-making processes can be explained and justified, allowing users, developers, and stakeholders to trust and verify the outcomes. Explainable AI is critical in applications where decisions impact lives, such as healthcare, finance, law, and autonomous vehicles. While ML produces highly complex and seemingly accurate models, interpreting and verifying them with automated formal verification is problematic. This challenge led to the development of eXplainable AI (xAI) to formally support verification efforts.[19] In Figure 4.1 a taxonomy is shown of the different aspects according to which the ML models can be explained.

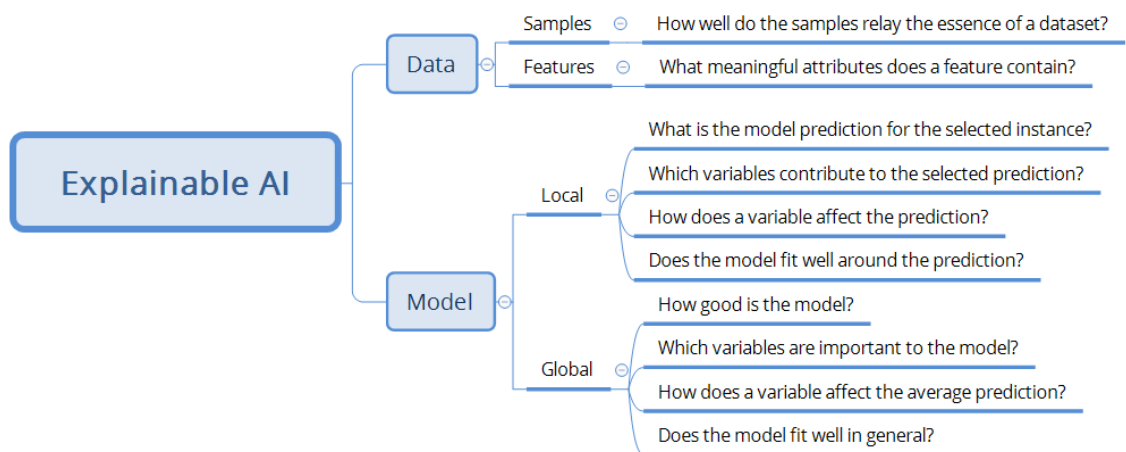


Figure 4.1: Explainable AI taxonomy

4.1 Explanatory Model Analysis

Before applying supervised learning to a dataset, it is crucial to understand its characteristics and features. However, single features in a dataset can sometimes combine multiple attributes, making them harder to interpret. The DIP-VAE [29] algorithm is an unsupervised representation learning method that takes existing features and learns a new representation where each feature is disentangled, making them more interpretable and meaningful to end users. ProtoDash [20] helps understand a dataset through prototypes, samples that capture the core characteristics of the dataset, and criticisms that are outlier samples, deviating from the general pattern.

As for model exploration methods, there are *interpretable-by-design* models. These are inherently designed to be easily understandable by humans, as their structure and outputs are straightforward. These models include decision trees, linear or logistic regression, or rule-based methods such as BRCC (see 4.2.2) or RIPPER (see 4.2.3). In addition, there are *model-agnostic methods* that can be applied to explain any type of ML model, regardless of its architecture (e.g. LIME [38], SHAP [30]), and *model-specific methods* that are explanation techniques designed for particular types of models, taking advantage of their unique structures or learning processes (e.g. feature importance in random forests).

Local explanations focus on interpreting the model’s behavior for individual instances, the following questions, which represent layers depending on how deep the explanation is:

1. **What is the model prediction for the selected instance?**
2. **Which variables contribute to the selected prediction?** (e.g. LIME, SHAP)
3. **How does a variable affect the prediction?** (e.g. *ceteris paribus*)
4. **Does the model fit well around the prediction?**

Global explanations aim to provide a broader view of the model’s behavior across the entire dataset, answering questions about overall performance and feature importance:

1. **How good is the model?** (e.g. ROC curve)
2. **Which variables are important to the model?** (e.g. permutational variable importance)
3. **How does a variable affect the average prediction?** (e.g. partial dependence profile)
4. **Does the model fit well in general?** (e.g. residual-diagnostic plots)

Popular xAI frameworks are IBM AIX360[1], DALEX[5, 3] and Intel Explainable AI tools[23].

4.2 Typical xAI models

Building interpretable-by-design models manifests in making models interpretable step-by-step, sacrificing some accuracy but retaining a good approximation of the data. Their essence lies in constructing models by combining simple, elementary steps, which enhances interpretability without fully compromising performance.

4.2.1 Decision Tree

The decision tree [37] (DT) is one of the simplest and most popular ML algorithms created in the early days. Although it is simple, it is surprisingly accurate and very versatile. It can be perfectly used to solve both classification and regression problems.

This model is represented by a tree graph $T = (V_{tree}, E_{tree})$ so that $E_{tree} \subset V_{tree} \times V_{tree}$. The tree's root is represented by $root \in V_{tree}$. The idea behind it is to recursively split the data set into smaller subsets until the stopping condition is satisfied. This condition could be a depth constraint or even that a subset contains only one type of label. The splitting is performed as follows: in each recursive step, the best split is searched for. This can be done by trying to split along all attributes, including all values. Splitting a dataset $D = \{(x, y)^n\}$, where $x \in X_{data}$ and $y \in Y_{data}$ along the feature *feature* and value *val* means, that we create two subsets X_1, X_2 and Y_1, Y_2 of each X_{data} and Y_{data} , so that samples with values of *feature* less than *val* belong to X_1 , otherwise they belong to X_2 . Y_1, Y_2 contain the labels associated with X_1, X_2 . In other words, $X_1 = \{x | x \in X_{data} \wedge x[feature] < val\}$ and $X_2 = X_{data} \setminus X_1$. The best of these is selected, which is done by measuring the impurity of the subsets. The best split will be the one where the weighted impurity of the two subsets is the smallest since this is where the information gain is the largest. There are different metrics for measuring impurity, such as entropy or Gini impurity. When we have reached the depth limit, our subsets have only one type of label, or we cannot split any further for some other reason, we create a leaf. This will have a label that, when estimated, will traverse the tree and give the result of the decision.

This way, a tree graph is built: its vertices represent the *split conditions*, its edges represent the true and false branches of the conditions, and its leaves represent its predictions.

A split condition of node $v \in V_{tree}$ looks like $splitcondition(v) := feature(v) < val(v)$. Let v_{true}, v_{false} be v 's children, the roots for the subtrees that contain samples split by v 's split condition. This way the edge (v, v_{true}) has the *edge condition* of $edgecondition(v, v_{true}) := feature(v) < val(v)$ and the edge (v, v_{false}) has an edge condition of $edgecondition(v, v_{false}) := feature(v) \geq val(v)$. A *path condition* of a node v is the conjunction of each edge condition from *root* to v , i.e. $pathcondition(v) = edgecondition(root, v_1) \wedge edgecondition(v_1, v_2) \wedge \dots \wedge edgecondition(v_i, v)$.

A *ruleset* is a set of rules that describe the decision process of the model when it predicts a positive label. In DTs, this can be obtained by traversing the tree and taking the disjunction of the path conditions of all leaves.

$$ruleset(T) = \bigvee_{\substack{v_i \in V_{tree} \\ deg(v_i)=1}} pathcondition(v_i)$$

In this research, the implementation of DT from the scikit-learn [36] library was used. It is worth mentioning *Random forest classifiers*, which are ensemble machine learning models that use multiple DT to improve classification accuracy by averaging their predictions and reducing overfitting.

4.2.2 Boolean Rules via Column Generation

The Boolean Rules via Column Generation (BRCG) algorithm[11], implemented in the AIX360 toolkit[1], is a very efficient way to generate CNF or DNF rules from a data set, that solves the problem of learning Boolean decision rules by formulating it as a linear

program. The algorithm tries to minimize the number of false positive and false negative classifications.

Given a training dataset $D = \{(x, y)^n\}$, that can be partitioned into $P \cup Z$, where P and Z contains the indices of the positive and negative samples. Let K denote the collection of all possible clauses to be used in the ruleset, and K_i denote the clauses satisfied by sample i . $\xi_i \in \{0, 1\}$ denotes if sample i was misclassified, w_k denotes whether clause k was used in the ruleset, and c_k denotes the complexity of clause k . C is a parameter that bounds the complexity of the ruleset to prevent overfitting. The *master integer program* (MIP) is the following:

$$z_{MIP} = \min \sum_{i \in P} \xi_i + \sum_{i \in Z} \sum_{k \in K_i} w_k \quad (4.1)$$

$$\text{s.t.} \quad \xi_i + \sum_{k \in K_i} w_k \geq 1, \quad \xi_i \geq 0, \quad i \in P \quad (4.2)$$

$$\sum_{k \in K} c_k w_k \leq C \quad (4.3)$$

$$w_k \in \{0, 1\}, \quad k \in K. \quad (4.4)$$

The objective function (4.1) has two components. The first component it tries to minimize is the number of misclassified positive samples, i.e., the false negative samples. The second component to be minimized represents the number of clauses that satisfy negative samples, in other words, the number of clauses that can lead to false positive predictions. The first constraint (4.2) says that all positive samples should be covered by clauses otherwise they are incorrectly classified. Basically, it identifies false negative predictions. The third constraint (4.3) bounds the complexity of the ruleset.

However, this MIP is not practical with real-life datasets, as there are exponentially many possible clauses. Therefore, solving it would be computationally impossible. That is why it is solved using the *column generation* (CG) approach. This means, as an initial state, there is a restricted master linear problem (RMLP), which is a linear programming (LP) relaxation of the MIP where the set K containing all the possible clauses is replaced with a very small subset of it containing simple clauses. As an initial step, the RMLP is solved. After that, a *pricing problem* is formulated, which is another IP that searches for a new clause (column, hence the name CG) that, if added to the RMLP, would improve the objective function of the RMLP the most. The metric for this is *negative reduced cost*. In each iteration, the pricing problem searches for the clause with minimal negative reduced cost, which is added to the RMLP and solved again, and the whole process is repeated. If there is no additional clause with a negative reduced cost, the RMLP is optimal, and the final ruleset is done.

4.2.3 Repeated Incremental Pruning to Produce Error Reduction

The Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [9] algorithm is a rule-based classification algorithm that uses the separate-and-conquer approach to generate rules for classification tasks. It is an efficient, incremental approach that is optimized particularly for large, noisy datasets and performs well with imbalanced class distributions.

It works in several stages, creating rules for each class in a one-vs-all approach, where one class is separated from all others.

1. **Initialization:** The classes are processed in increasing order of occurrence, so it finds the minority samples and selects them first.
2. **Growing phase:** It generates rules iteratively for a specific class until no more examples of that class remain uncovered. For each rule R , it starts with an empty rule and iteratively adds conditions (conjunctions) to the rule. Each condition c is selected to maximize the information gain, which can be computed as $\Delta gain(c) = gain(R_i \wedge C) - gain(R_i)$, where R_i is the current rule in an iteration. This algorithm uses FOIL's information gain as a metric, which is a heuristic measure. It stops adding new conditions when the rule starts covering the negative samples.
3. **Pruning phase:** After each rule is grown, RIPPER prunes it to avoid overfitting. For a rule R_i with n conditions, it is evaluated after removing each condition in turn, aiming to maximize the error reduction. The evaluation is based on a pruning metric, which is $v(R, P) = \frac{p-n}{p+n}$, where P is the current pruning set. This value becomes 1 if the rule only covers positive samples and -1 if it only covers negative ones, so the more positive samples are covered, the better.
4. **Optimization and ruleset replacement:** For each rule, there are three options. They can be kept, modified, or replaced with a completely new rule. This selection is based on a metric called Minimum Descriptive Length (MDL), which combines rule complexity and errors. It balances between rule complexity and accuracy.

4.2.4 Interpretable Model Differencing

The Interpretable Model Differencing (IMD) [21] algorithm is a very good method to show differences between two machine learning models in an interpretable way. It is based on building a Joint Surrogate Tree (JST), which is a novel representation for replacing the two models being compared. From this, it can be read where the models agree and differ, and also, a ruleset can be extracted that describes where the two models contradict each other.

The first step is to create the two models $M_1, M_2 : X_{data} \mapsto Y_{data}$ from the input data set. We want to examine the differences between them. Next, we need to evaluate the features of the input dataset on the created models to obtain Y_{M_1}, Y_{M_2} , the labels of the two models. Then we train an IMD instance, which is a binary classifier $imd : X_{data} \times Y_{M_1} \times Y_{M_2} \mapsto Y_{imd}$, which takes the features from the dataset and the prediction of the two models as the input and predicts if the two models contradict to a given sample.

This model works by building a JST (Figure 4.2) based on the data, which is like a decision tree. It is a tree graph containing decision nodes and also special nodes called *OR-nodes*. Let $JST = (V = V_{fa} \cup V_{OR}, E = E_{fa} \cup E_{OR})$ where V_{tree} marks the set of the decision nodes, V_{OR} marks the set of the OR-nodes. E_{OR} contains all the edges that connect OR-nodes with their children. This is important because OR-nodes mark the *divergence* of the two models, which means at this point if the two models were replaced with two surrogate decision trees, they would not have the same split condition at that node. The nodes from V_{tree} can be *common nodes* where the two surrogate decision trees would have the same split conditions, or they can be separate nodes belonging to the surrogate decision trees of the two models. Note that a subtree of an OR-node can not contain common nodes anymore. The JST also contains leaves containing the models' labels that can be either

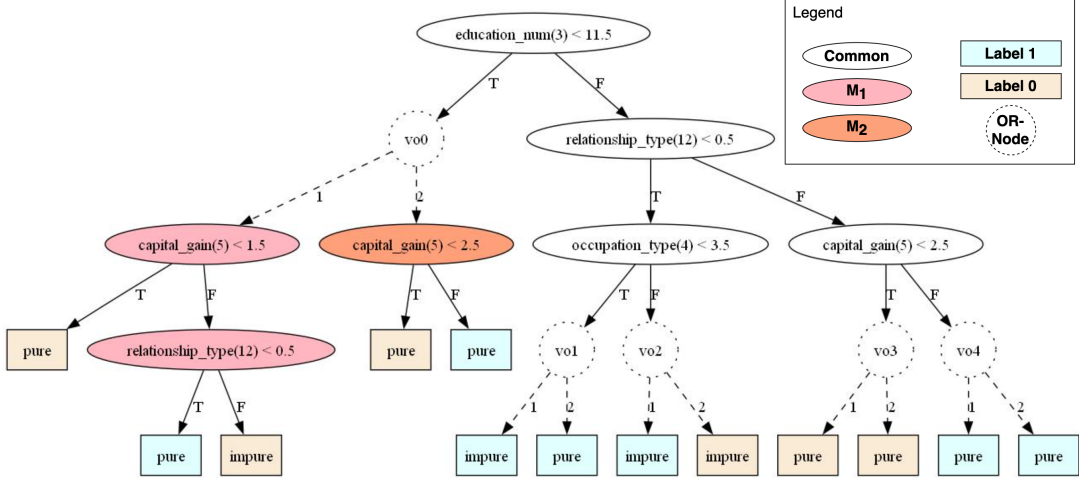


Figure 4.2: Joint Surrogate Tree

pure or *impure*. The impure leaves contain both positive and negative labels. The final prediction is the majority label. This is because of the depth constraint, so impure leaves can be further refined.

A JST is built recursively. Starting from the root, at the creation of each node it must be decided whether to create a common decision node or an OR-node. Once the divergence has happened, the two subtrees no longer share split conditions, and the creation of nodes is done exactly as when building a simple decision tree. The decision of whether to create a common node or an OR-node is based on comparing the impurity of the best possible common split to the sum of the best possible individual splits. Let $H(val, feature, X_{data}, Y_{data})$ be the weighted entropy of the two subsets of labels cut upon the feature *feature* and value *val*. In this case, the impurity of the best split is:

$$impurity(X_{data}, Y_{data}) = \min_{\{feature, val\}} H(val, feature, X_{data}, Y_{data})$$

The impurity for the best common split is:

$$impurity(X_{data}, Y_{data}) = \min_{\{feature, val\}} H(val, feature, X_{data}, Y_{M_1}) + H(val, feature, X_{data}, Y_{M_2})$$

In this case, an OR-node is created when the following condition containing $\alpha \leq 1$ parameter fulfills:

$$impurity(X_{data}, Y_{M_1}) + impurity(X_{data}, Y_{M_2}) \leq \alpha \cdot impurity(X_{data}, Y_{M_1}, Y_{M_2})$$

A difference rule can be extracted from a JST by selecting an OR-node, two leaves from its two subtrees with different labels, and conjunction the path conditions of these leaves.

$$diffrule = \{pathcondition(leaf_1) \wedge pathcondition(leaf_2) : leaf_i \in leaves(vo_{OR}), i = 1, 2 \\ label(leaf_1) \neq label(leaf_2)\}$$

4.3 Qualitative modelling in xAI

Qualitative modeling in xAI seeks to enhance the interpretability and transparency of AI systems, allowing for a clearer understanding of how these systems arrive at their decisions. QM can be applied in several ways to provide high-level, human-understandable explanations.

One way of applying qualitative modeling in XAI is to create *concept-based explanations*, where complex patterns are distilled into high-level concepts and their relationships. This means generating explanations that align with human knowledge frameworks, often organized in ontologies, which map and structure these concepts and can aid in knowledge creation. Through *causal reasoning*, qualitative models also explain cause-effect relationships, enabling users to understand how different factors influence an outcome. Qualitative abstraction plays a key role by allowing complex systems to be represented in simpler terms through surrogate models, which approximate the behavior of more intricate algorithms without compromising accuracy.

To account for the inherent uncertainty in AI systems, qualitative modeling can also be used for *uncertainty representation*. They can represent uncertainty more intuitively and allow the usage of methods like rough set theory or sensitivity analysis. The usage of rule-based systems can further simplify transparency by converting decision rules into qualitative models through predicate abstraction, making them more interpretable. In addition to logical reasoning, analogical and temporal reasoning can also enhance interpretability. Analogical reasoning uses analogical explanations to help users intuitively grasp the model’s behavior, while temporal reasoning enables the understanding of time-dependent processes.

Most interpretable-by-design models allow the extraction of decision rulesets, which are then can be converted into qualitative models, providing an abstract representation of their decision processes for further analysis.

4.4 Ruleset to qualitative model

The core of xAI modeling is the composition of simple rules into such a fabric, which sufficiently faithfully approximates the dataset. The core idea of my work is my observation that the most popular models all use some combinations of simple inequalities to partition the different behavioral domains waived by logical formulae. This way transforming the inequalities to qualitative variables is a general principle to gain the qualitative counterpart models. In more precise form, I realized that all of their explanations can be represented with a tree graph. In this graph, every node represents a decision rule in the form of split conditions which are inequalities. These inequalities can be transformed directly in a qualitative model.

Each value in these inequalities in a ruleset represents a *qualitative landmark*. This means the continuous domains of the features can be clustered by partitioning them into intervals using interval logic [28] along the split values. After that, each partition is assigned a *qualitative value*, which is named using domain knowledge. This process is called *qualitative abstraction*. Let $feature \in \mathbb{R}_+$ be a feature and R be a simple rule containing one split condition: $R = \{feature < val\}$. In this case, the domain can be partitioned into $I_1 = (-\inf, val)$ and $I_2 = [val, +\inf)$. To these intervals, q_1 and q_2 qualitative values can be assigned.

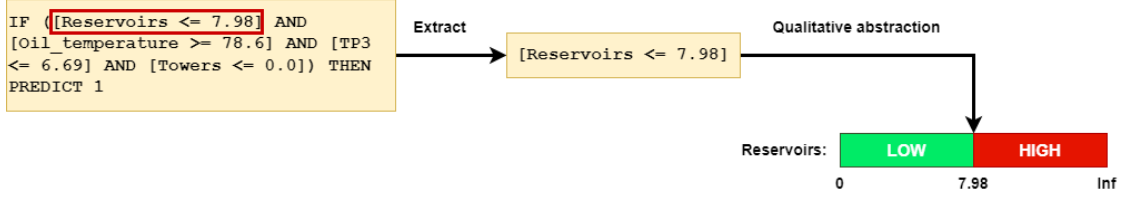


Figure 4.3: Qualitative abstraction

As an example, in a ruleset built by training a rule-based model on the MetroPT-3 dataset[12], there is a condition: $[Reservoirs \leq 7.98]$. After qualitative abstraction using the process described above, we can say, that the *Reservoirs* feature could have two qualitative values: *LOW* and *HIGH*. This process is shown in Figure 4.3.

Converting a ruleset into a qualitative model means, we apply this process to all of the variables using all of the conditions in the ruleset. The clustering of overlapping conditions is solved using interval logic.

Chapter 5

Impact Analysis of ML models

In CPSs, embedded AI models mainly serve the function of deciding whether to intervene or not in a process. They usually consider different factors such as the system's states or other circumstances. More specifically, embedded AI's two main functions in critical systems are fault detection and diagnosis or real-time control. For instance, a diagnostic model can monitor engine performance in an aircraft, detect anomalies in medical devices, or identify potential failures in industrial equipment. In real-time control systems, this can be used for autonomous vehicle navigation for industrial robot operation. Possible interventions in the above implementations could be: rerouting to the nearest suitable airport on dropped engine performance, emergency stop procedures in industrial equipment, or breaking when detecting a passenger on the road. For all of these situations, a missed intervention is indeed more costly than an unnecessary intervention.

This research report has been focusing on binary classifiers used for failure prediction (Figure 5.1), but all of the challenges and approaches it presents apply to the situations discussed above. In this context, a binary classifier is trained on a dataset, which contains *observations* i.e. diagnostic values (e.g. sensor values, states of actuators) generated by a component in a CPS. This classifier predicts a positive label if the component fails and an *alert* should be made about it, and a negative one if it is operating properly.

Let X_{diag} be the input feature space, θ the model parameters and $y \in \{0, 1\}$ be the binary output where:

- $y = 1$ represents an alert
- $y = 0$ represents normal operation

The classifier aims to learn the function $predict : X_{diag} \times \theta \mapsto Y_{diag}$, where $Y_{diag} = \{0, 1\}$.



Figure 5.1: Failure Prediction

5.1 Challenges

The following two subsections will outline some of the main challenges associated with using embedded AI models in critical systems such as failure prediction.

5.1.1 Different impact of FP and FN predictions

False negative and false positive samples having a different magnitude of impact on the system is one of the key issues I examined. In more detail, false negative samples, which lead to unpredicted alerts, can produce high level of hazards on the system's output. In contrast, false positive predictions often represent false alarms which mostly lead to reduced availability but maintained stability. The use of an asymmetrical cost function when creating the model is convenient for this reason. The idea behind this is to build a more sensitive model, which has a high chance of predicting alerts and it's not a significant concern if the false alarm rate is higher. Our objective is to reliably predict a component malfunction when it appears i.e. when an alert should be made, with a high degree of certainty.

An ideal model for this application would have the following cost function. Let y be an expected prediction and $\hat{y}$ be an actual prediction. An ideal loss function for this looks like the following:

$$L(y, \hat{y}) = \begin{cases} \alpha & \text{if } y = 1 \text{ and } \hat{y} = 0 \text{ (False Negative)} \\ \beta & \text{if } y = 0 \text{ and } \hat{y} = 1 \text{ (False Positive)} \\ 0 & \text{if } y = \hat{y} \text{ (Correct Prediction)} \end{cases}$$

The cost function is the expected loss of the model:

$$R(\theta) = \mathbb{E}[L(Y_{diag}, predict(X_{diag}; \theta))] = \sum_{i=1}^n L(y_i, predict(x_i; \theta))P(x_i, y_i) = \alpha \cdot P(FN) + \beta P(FP)$$

To build a model that is more strict towards the FN values, an asymmetrical cost function should be used where: $\alpha > \beta$.

5.1.2 Highly imbalanced dataset

In data science and machine learning, dealing with unbalanced datasets is a big challenge. This happens when the data is heavily skewed toward one class over the other. This imbalance makes it hard for models to learn about the minority class, often resulting in biased predictions and poor performance. This imbalance is common in critical applications, where the datasets are highly imbalanced, because, in most real-world scenarios, faults or rare events do not happen as often as normal or expected behavior.

5.2 Error Propagation Analysis

The core objective of *Error Propagation Analysis* (EPA) [27, 33, 15] is to assess how localized faults within a system can cascade and ultimately impact the overall system dependability. This method is essential for identifying how component-level faults and fault combinations can cause critical consequences on the system's output by which requirement violations are meant.

EPA is based on an *error propagation model*, which is an abstract representation of system models that include components, their behavior (as discrete propagation rules), resource usages, and data/information flow between components. The EPA model can be con-

structured from the system’s structural model and augmented with abstract operational dynamics derived through engineering interpretation. The potential paths for error propagation are determined by the connections that describe the data and information flow between components. In my approach, I use an *abstract qualitative model* (see figure 5.2) of the system as an EPA model.

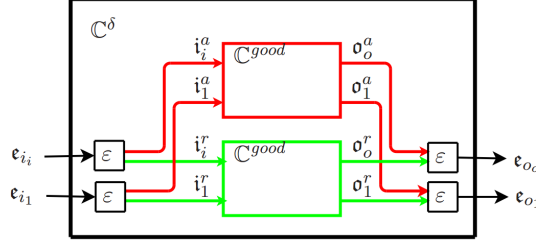


Figure 5.2: Qualitative EPA model

Propagation rules represent component behaviors and describe how errors propagate from one component to another. Two EPA modeling approaches are available: *value* and *property propagation*. The value propagation approach provides insight into how changes in component values can impact system operations. In this approach, components are represented using qualitative models. For instance, continuous values are converted into qualitative values. The evaluation process involves analyzing these qualitative values to systematically assess how faults in the values may affect the overall system behavior. The property propagation approach focuses on analyzing the impact of specific component property violations (which often represent attributes of dependability), such as integrity faults. This approach examines how these properties propagate through the system, affecting other properties along the propagation path. It determines how the violation of properties can affect the overall dependability of the system.

From the component point of view, in EPA, there are two types of propagation: *internal* and *external propagation*. Internal propagation means error propagation within a component is caused by an internal computation process, i.e., an error successively transforms into other errors inside a component. External propagation occurs when, through internal propagation, an error reaches the service interface of a component, thus propagating into another component the first one is connected to potentially triggering failures in the receiving component. The EPA approach used in this research focuses on external propagation within a system.

5.3 Context-independent model analysis

The context-independent model analysis consists of examining the model in itself disregarding the context the model is currently operating in. The goal of this analysis is to find out how to increase model accuracy or lower the ratio of false negative predictions. In this approach the performance of the model these measured by metrics such as accuracy, precision, recall, or the area under the ROC curve. The improvement of these metrics can be approached from the *dataset side* or the *model side*.

5.3.1 Dataset-side improvement

From the dataset side, preprocessing could be done to increase the efficiency of the training process. This may include the following steps:

1) **Handling Missing Values:** Samples containing missing values are dropped or missing data is filled using simple values (e.g. default value, mean, median) or advanced imputation techniques. 2) **Removing Outliers:** Outliers, which may introduce distortion, are identified and removed. 3) **Encoding Categorical Values:** Categorical variables are transformed into numbers using methods like one-hot or label encoding. 4) **Normalizing Data:** Scaling data (using e.g., min-max, z-score) ensures all features contribute equally, critical for algorithms sensitive to feature magnitude. 5) **Data Transformation:** Transformations like logarithmic scaling reduce asymmetry in data distributions making them more suitable for learning. 6) **Feature Selection:** Removing correlated features prevents redundancy, or calculating feature importance and selecting a lower number of features that most affect the target variable improves efficiency. 7) **Adding Semantic Redundancy:** Domain knowledge is used to create or enhance features, improving model performance by adding meaningful data.

As an experiment (see A.3.1), I trained a decision tree classifier on the adult data set to predict the wealth of people. This data set contains whether a person’s income exceeds \$50k based on demographic and employment features. Then I quantized some features based on domain knowledge, added them back as a redundancy, and compared the two models. In my case, the model’s accuracy did not increase but the decision tree split upon our newly added features instead of the original ones.

As another experiment (see A.5.1), I trained two neural network models with the exact same parameters on the MetroPT-3[12] dataset, one with preprocessed data and one with raw data. The preprocessing consisted of removing outliers and subsampling. The one trained with preprocessed data produced a 3% higher precision, 5% higher recall, and a 4% higher f1-score on positive labels. As the normalization of numerical features before training a neural net is a necessary step, I did not omit it when training the model with raw data. Otherwise, the precision would drop by 22%, the recall by 13% compared to the model trained on preprocessed data.

5.3.2 Preprocessing imbalanced datasets

Possible Solutions for processing this high level of imbalance include subsampling, oversampling, and Synthetic Minority Oversampling Technique(SMOTE). Subsampling means we remove samples from the majority class until we get an equal distribution i.e. a balanced data set. In other words, we separate the dataset into S_{min} containing the minority samples and S_{maj} containing the majority ones. $S_{sub} \subset S_{maj}$ contains fewer samples from the majority class so that $|S_{sub}| = |S_{min}|$. The new dataset is $D = S_{min} \cup S_{sub}$. This new data set is an abstraction of the original one containing fewer data points. Because the number of observations in this case is limited, it is far from certain that the data has the necessary representativity. This means the chances to learn the faulty cases are slim.

On the other hand, oversampling consists of multiplying the samples of the minority class until the class distribution is equal. The new dataset is $D = S_{maj} \cup S_{over}$ where $S_{over} \supset S_{min}$ contains repeated samples from S_{min} . This can introduce a huge distortion and the faulty cases, which are originally represented by the minority class will be far overrepresented. This can cause overfitting and because there are many types of faults and each of them has too few appearances in the dataset, we cannot say much about them.

An upgraded version would be SMOTE which creates synthetic samples based on the minority class. It does this by in each iteration selecting a sample from the minority class and searching its k nearest neighbors. The synthetic samples are made by interpolating between the selected sample and its nearest neighbors. Although this is a more sophisticated method, which contributes more to the accuracy of the model, it still has the same problems as oversampling i.e. over-representativity and distortion.

5.3.3 Model-side improvement

From the model side, improvements can be made by tuning the different parameters of the model. A prime example of this is *cost-sensitive learning*. This not only helps to learn imbalanced data sets but is also useful for the creation of a sensitive model, which is convenient in the context of critical systems. It consists of changing the cost function while training the model according to the risks. For example, to the positive class way higher weights are assigned in a DT, than to the negative class. In my experimentation, I trained the DT on the MetroPT-3 data set [12] by altering the class weights so that the positive class got a magnitude higher rate than the negative one. (see A.5.3) This resulted in a higher sensitivity which covered more alerts than the one trained with equal class weights. A relatively high rate of false alarms because of lower specificity is the trade-off of this method. The problem with this approach is that the model can produce unpredictable or inconsistent outputs, which cannot be used for deriving meaningful insights. In other words, the model will be unstable and it might struggle to generalize properly, causing its outputs to be inconsistent. It can also have a low selectivity which means it tends to over-generalize and it has difficulty distinguishing between the positive and negative samples.

There are also *boosting methods* that combine dataset side and model site approaches. This is an ensemble learning technique that iteratively combines multiple weak learners to create a strong predictive model. In this process, each subsequent learner focuses on correcting the errors of previous ones. For instance, AdaBoost [17] increases sample weights in each iteration so that each new learner focuses more on the previously misclassified samples, then the final prediction will be the weighted average of the weak learners' results.

5.4 Context-dependent model analysis

My focus for a model is not on performing well in general but in the given context, i.e., the actual system it is embedded in. We want to ensure the dependability of the system by identifying *critical cases* (Fig. 5.3) such as component-level faults that lead to the violation of safety or security requirements. In my work, I am mainly focusing on identifying the critical points of the embedded AI model. These points represent faulty behavior that leads to the failure of the system making it unusable. My goal is to perform an exhaustive analysis where I examine all possible local faults of the model and evaluate their impacts on the system's output. Using this information I can build a model that performs well in the given context, in other words, minimizing in the design time the hazards it can cause.

This process is called *impact analysis*. It is a systematic assessment of how changes in the behavior of one component or module may affect the output of the system. It effectively assesses how sensitive is the system to various faults of components. This sensitivity analysis is evaluated using efficient discrete formal methods.

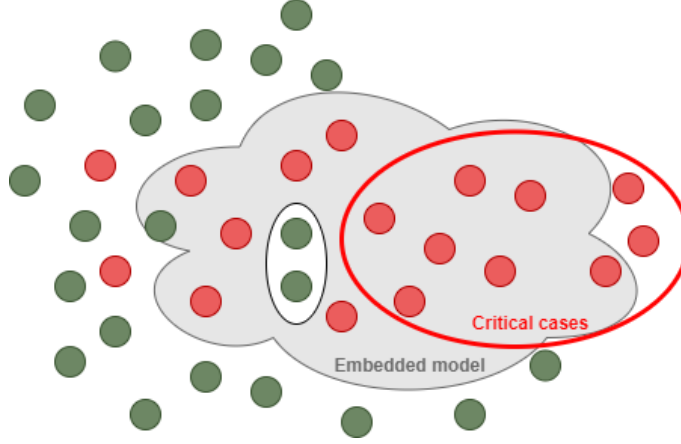


Figure 5.3: Critical cases

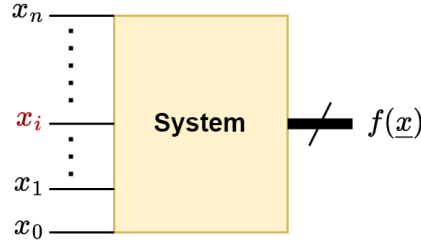


Figure 5.4: System as logic function

For the sake of simplicity, let the logic function $f : \{0, 1\}^n \mapsto \{0, 1\}$ represent our system model as shown in Figure 5.4. My aim is to examine, how sensitive is the system to the change of an input variable. In other words, how $f(\underline{x})$'s value changes as a function of x_i . This can be calculated by trying out every possible value of x_i and examining the value of the function. In other words, by calculating $\frac{df}{dx_i}$. This can be done by using boolean differentiation[39] in the following way:

$$\frac{df}{dx_i} = f(x_0, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \oplus f(x_0, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) = f_{0,i} \oplus f_{1,i}$$

As the values of the other variables are not fixed, calculating $f_{0,i} \oplus f_{1,i}$ ultimately reduces to solving a SAT instance, so it is NP-complete. Because of this, in practice, discrete formal methods are used to perform impact analysis.

In my research, I used EPA for impact analysis. (See 5.2) The qualitative reasoning-based EPA operating on an abstract model, where continuous variable domains are clustered into qualitative states, opens the rich repertoire of discrete formal methods, such as model checking for dynamic requirements. In *context-dependent model analysis*, we determine the faults of the model such as unpredicted alerts, build a qualitative model of these cases, and propagate these errors to the system's output using EPA. This way we know which cases are critical or not. Finally, based on this evaluation, we improve the performance of the embedded AI component.

5.4.1 Faults projected forward

This approach consists of analyzing the behavior of the embedded model and projecting its faults forward onto the system's output. Then it is analyzed whether a fault has critical consequences or not. This impact analysis is realized by *EPA*.

The first method of model analysis consists of using a transparent, *white-box model* in production so that its behavior can be easily analyzed (architecture shown in Figure 5.5). Explainable AI models are best suited for this because their explanations can be directly converted into *qualitative models*. These describe the behavior of the AI component so they can be used for the evaluation of EPA.

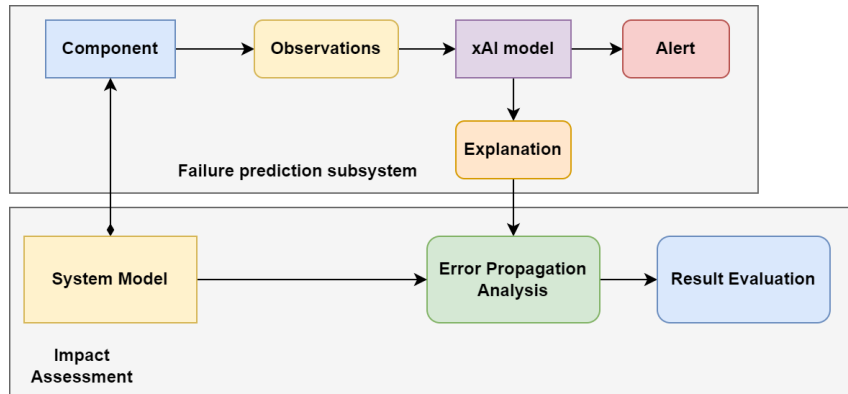


Figure 5.5: Using xAI model

In the simplest case, we have only the positive samples, which describe a scenario where an intervention is required. For instance, samples describing cases when an alert should be made about a failure or anomaly in the system. For these situations, a good practice is applying a *one-class classifier* [32] model. This type of classifier detects outliers efficiently so it is a widely-used technique for novelty or anomaly detection. It consists of making a model that describes objects of a specific class by primary learning from a training set containing only the objects of that class. This approach includes density estimation, boundary creation, or reconstruction methods. For instance, we warp a convex hull around the data points representing the positive class. The nested cavities algorithm [22] is an example of making this model, using only the positive samples. This is an iterative algorithm that encloses a subset of the target class in each iteration, while allowing some misclassifications (false positives or false negatives), refining the sample space until termination criteria are met. This model can be quantized so that a qualitative model can be created that describes the cases the positive samples represent. By evaluating EPA on this qualitative model we get which of these described cases are critical or not. For example, by quantizing the boundary of the convex hull a proper qualitative model can be made for evaluating impact analysis using EPA. The biggest drawback of this approach is that there could be other cases when an alert should be made, that are not represented by the positive samples in our data set. For instance, considering the convex hull, data points describing these cases could also exist outside the hull, just because they are not contained in the dataset, rather than belonging to the other class.

Another method, when we also have negative samples that describe situations when an intervention is not required such as samples describing a normal operation for failure prediction, is to use a transparent model such as an interpretable-by-design xAI model. From these models, a qualitative model describing its behavior can be easily extracted. Based on this qualitative model sensitivity analysis can be conducted using EPA. Examples

of these xAI models are DTs, BRCC, or RIPPER. In my experiment, I trained these three models on the MetroPT-3[12] data set and extracted a set of decision rules from them. These rulesets describe how the models classify between positive and negative samples. They contain rules which are a set of split conditions in CNF or DNF form. From these rules, I created the qualitative model using interval logic. From this qualitative model, I created error propagation rules for the embedded AI component for evaluating EPA on the system.

5.4.2 Difference modeling

A more sophisticated approach is to use redundant embedded models in the system. (Figure 5.6) These two models are trained on the same data set and have the exact same function. The idea is that if their predictions correspond, then there's a high chance that they predicted correctly. Note that there's also a possibility that both of their predictions are incorrect, but the chances for this are slim. If they contradict each other, it marks that the current case is inside a region of uncertainty. If we place the samples in an n -dimensional space this represents the boundary region between the positive and negative ones. This is the region that represents edge cases and most of the mispredicted samples are found here. In these situations, rough set theory can also be applied. [34, 40]. A *difference model* is an xAI model that describes the difference between two models. It predicts dissimilarity between two models while also being interpretable. This type of model is used to describe the region of uncertainty.

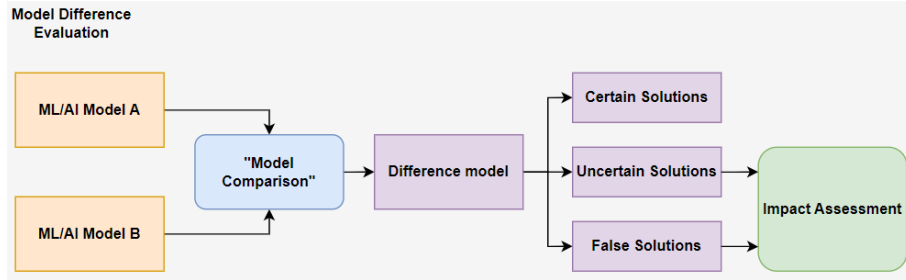


Figure 5.6: Difference Model Evaluation

Let $M_1, M_2 : X_{diag} \mapsto Y_{diag}$ be two predictive models and a *dissimilarity function* $D : Y_{diag} \times Y_{diag} \mapsto \mathbb{R}_+$ (where $\mathbb{R}_+$ means the non-negative reals including zero), which measures the difference between the output of the two models (in this case Hamming-distance). The difference model is an interpretable-by-design model that learns the following mapping: $\hat{D} : X_{diag} \mapsto \mathbb{R}_+$ that predicts the dissimilarity $y = D(M_1(x), M_2(x))$.

The first type of difference model is a binary classifier which predicts 1 if the two examined models predict differently on the given sample or 0 otherwise. More formally, its dissimilarity function is: $D(M_1(x), M_2(x)) = M_1(x) \oplus M_2(x)$, where $\oplus$ is the logic *XOR* function and logic values *true* and *false* correspond to 1 and 0. This classifier is implemented by an interpretable-by-design xAI model so that a qualitative model can be built from its explanation. This qualitative model can be used for sensitivity analysis. In my experiment (A.5.2.1), I trained a DT to examine the differences between a random forest and a neural net model. From this DT, a rule set containing split conditions can be extracted by traversing it, and from these rules, the split values can be extracted as qualitative landmarks. From these, a discrete qualitative model can be built.

Another type of difference model I experimented with (A.5.2.4), is by examining all of the 8 cases which consist of all variations of the original label, the prediction of the first model, and the prediction of the second model, i.e. all possible values of the set $\{y_i, M_1(x_i), M_2(x_i)\}$.

With the help of this difference model, we can categorize the samples by these values and examine their distributions. In my experiment, I compared a refined neural network model with an abstract DT with high sensitivity, trained on the same data set (MetroPT-3). By analyzing the mentioned cases we could conclude that the high percentage of true negative predictions of the two models means that both models are good for identifying the majority class. The cases where the DT produced false alarms confirm the fact that it is a more abstract model. The relatively high number of true positive labels on both models conclude that they are both able to recognize the majority of cases when an alert is needed. The cases of false alarms generated by both models reinforce the fact that they generalize similarly. Considering the edge cases, there was a higher number when the DT made an alert and the neural network did not than otherwise, which confirms that the abstract model covers more alert-triggering cases than the refined one.

For a more detailed analysis the data set can be labeled with these 8 cases and the multi-class xAI classifier can be trained with it. From this, a qualitative model can be extracted that describes all these cases and this can be used for further qualitative reasoning or other discrete formal methods.

Building a JST as the last type of difference model with the IMD algorithm is a highly transparent method that facilitates a granular understanding of model behavior and decision-making differences between complex ML models. (an example of JST is shown in Figure 5.7) This surrogate tree shows where the two models predict similarly and where they diverge. It also provides a rule set as an explanation which can be turned into a qualitative model. In my experiment, I compared a random forest and the neural network model with interpretable model differencing and it could be seen that the neural network is a more refined model with a complex decision process at an edge case.

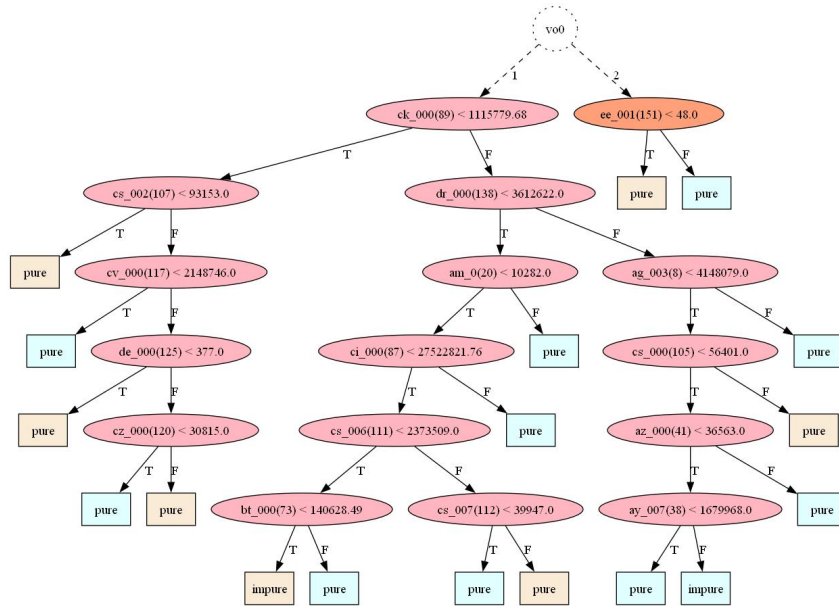


Figure 5.7: JST of NN vs RF model

By using an xAI model as a difference model a qualitative model from its explanations can be extracted and evaluated with EPA. The result of the EPA shows which of these uncertain cases are critical and worth more attention or not. In my experiments, I used DT, BRCC, and RIPPER to build a binary classifier on the contradictions of the two models, and also built a joint surrogate tree using the IMD algorithm. (see A.5.2.1) From all of these models, a rule set can be extracted which describes the uncertain cases. I turned these into a qualitative model to be evaluated with EPA.

5.4.3 xAI Surrogate model

Most of the time advanced *black-box models* are used in production. These offer superior performance as they often achieve better accuracy and performance compared to simpler, more interpretable models. They have the ability to handle complexity as they can automatically learn intricate patterns such as nonlinear relationships in high-dimensional data. However, their biggest disadvantage is that it is difficult to understand how decisions are made and they are hard to debug when errors occur, due to their complexity. An example of this type of model is a neural network which has a lot of fine-tuned parameters and is trained on clean, standardized data. My experiments used a neural network with two hidden layers trained on preprocessed data. This preprocessing consisted of handling missing values, outlier removal, subsampling, and standardization.

The analysis of these types of models is not as straightforward as the transparent models detailed above due to their lack of interpretability. My approach is to approximate their behavior with a simpler, easily interpretable model. (Figure 5.8) This idea originates from a formal method called *abstract interpretation* [10]. This is a theory of program analysis that approximates a program's behavior by mapping its concrete states to abstract domains. Since the data containing the observations lacks the necessary representation of expected behavior it should be approximated with an abstract model. This abstract interpretable model is called a *surrogate model*. This abstraction creates a Galois connection between the analyzed, highly refined model, and the surrogate model. This is a hybrid approach where a black-box model is explained in a model-agnostic (see 4) fashion using an approximation with an interpretable-by-design model.

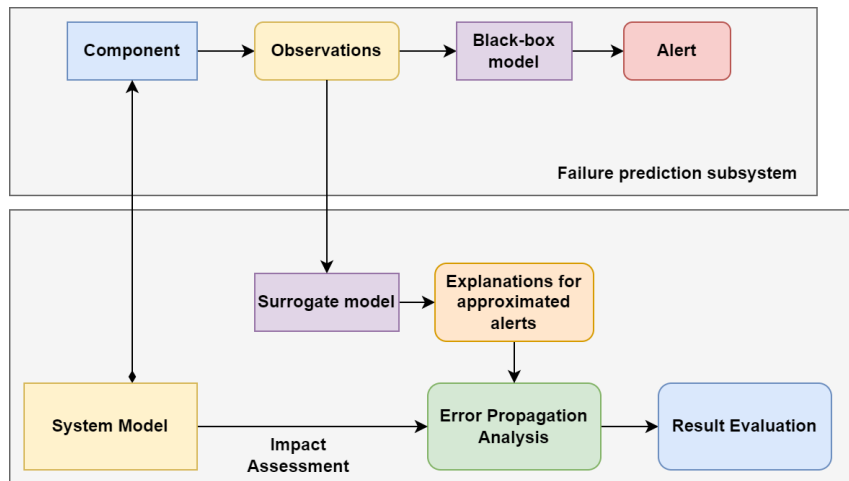


Figure 5.8: Application of surrogate model

The first method of approximating the behavior of the analyzed model is to build a model describing the positive samples. This can be achieved by using an explainable one-class

classifier as a surrogate model. As an example, the nested cavities algorithm builds a *convex hull* covering all positive samples in an n-dimensional space (Figure 5.9). This convex hull can be directly converted into a qualitative model on which EPA can be evaluated. The drawback of this method is that as the training data can be underrepresentative, it is not guaranteed that this model covers all possible cases such as predictions where alerts should be made.

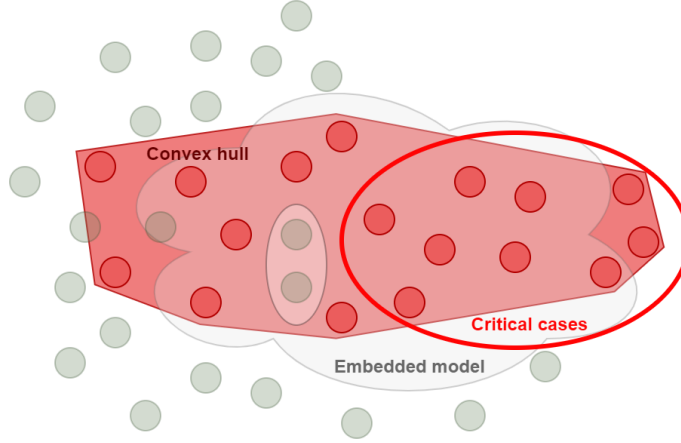


Figure 5.9: Convex hull covering samples

Another method is to use an *interpretable-by-design xAI model* as a surrogate model. Figure 5.10 shows how such a surrogate model covers all the positive samples in an n-dimensional space. The requirement for this model apart from interpretability is to be highly sensitive. It needs to be interpretable to be able to transform it into a qualitative model and sensitive to cover all positive samples. Covering 100% of these would imply a perfectly sensitive model with no false alarms yet, in practice this is often unachievable. Methods to build a highly sensitive model include oversampling, boosting, or cost-sensitive learning. In my experiments, I used a boosting method where in each iteration I trained a decision tree with unequal class weights and then increased the weight of misclassified samples.

As a further step of the analysis, a difference model can be built on top of the surrogate model and the one used in the system. This helps to identify the edge cases, which represent the boundary region between the positive and negative samples (Figure 5.11).

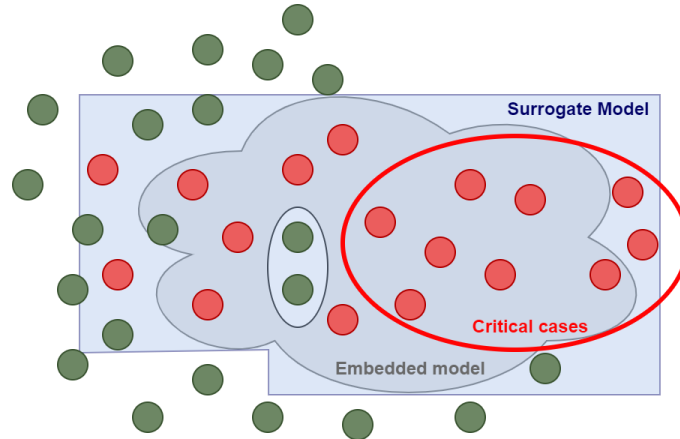


Figure 5.10: Surrogate model covering samples

This region contains the false alarms of the surrogate model and the unpredicted alerts of the embedded model.

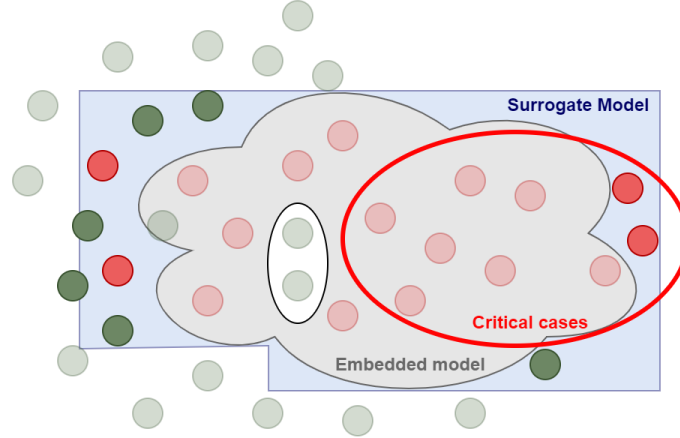


Figure 5.11: Boundary region

5.4.4 Faults projected backwards

The idea is to build a model which performs very well in its current application i.e. its predictions are the most accurate in the critical cases. This is achieved by determining those domains of the features using *combinatorial EPA* where it is critical for the system to react properly. Combinatorial EPA means we consider all possible fault activations of the system and evaluate whether they cause critical failures on the system's output or not. Based on these domains of features we customize the cost function of the model in a way to mispredict as few as possible in the *critical zones*. More mispredictions are allowed in *non-critical zones* where they are unable to cause as much problem.

My work aims not to refine the individual methods for improving embedded AI models but rather to develop a *universal* and *explainable* workflow for the evaluation and improvement of an embedded AI model. This workflow consists of the *evaluation of the critical cases*, *approximation of expected behavior*, *model evaluation*, and *model improvement*. Figure 5.12 shows the full workflow of model analysis.

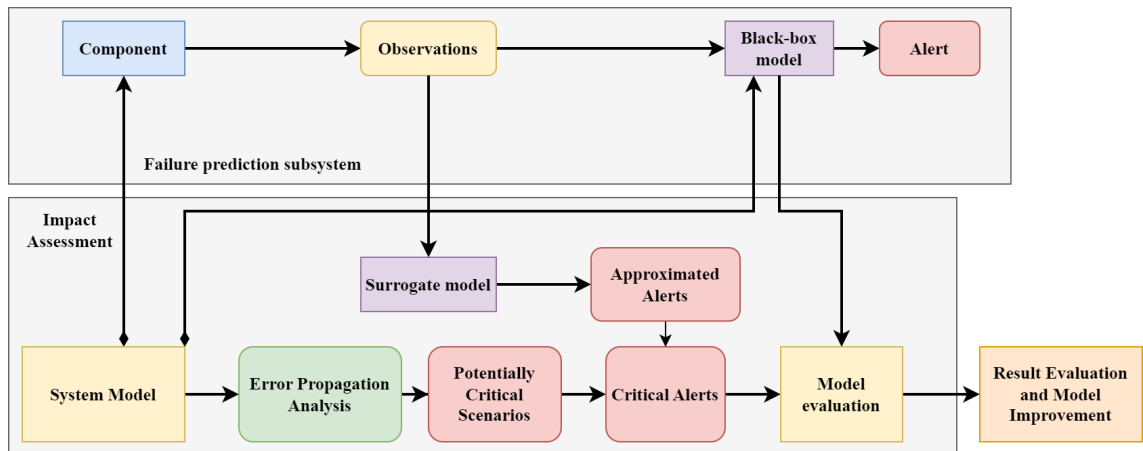


Figure 5.12: Workflow of analyzing the model backwards

The first step of the workflow is the *evaluation of critical cases* (Figure 5.13). Critical cases are defined as ones that lead to the violations of system-level requirements. These are requirements, such as safety and security requirements, without which the dependability of the system cannot be assured. This is achieved by creating a qualitative engineering model of the system. These system models include architecture data/information flow component behavior and fault models. On this engineering model, we evaluate combinatorial EPA to determine which qualitative values on its input lead to critical consequences.



Figure 5.13: Evaluation of critical cases

The second step is the *approximation of expected behavior*. (Figure 5.14) Approximation of expected behavior means building an abstract explainable model that describes the scenarios where an intervention is required, represented by the positive samples. As the model is an abstract one compared to the original black-box model, it represents an approximation.

For instance, those scenarios where an alert should be made about in a failure prediction context. This approximation is realized by building an *xAI surrogate model* on the top of the observations (as described in 5.4.3), which approximates the expected behavior in an interpretable manner. This means it gives an explanation to the behavior described by the observation, in our case, in the form of a ruleset.

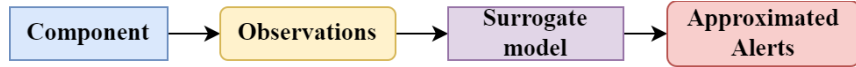


Figure 5.14: Approximation of expected behavior

The third step is determining *potentially critical system states*. (Figure 5.15) These are obtained by combining the critical scenarios that were evaluated by EPA with the approximations of expected produced by the surrogate model. In practice, this can be done by filtering the positive predictions of the surrogate model by the EPA results.

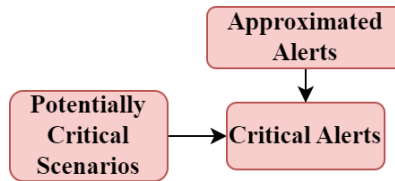


Figure 5.15: Potentially critical system states

Model evaluation (Figure 5.16) involves the calculation of the performance of the advanced production model by comparing its predictions to the potentially critical system states. This way we can obtain the non-detected critical cases represented by false negative samples. These can be examined by analyzing their distribution and also qualitative model can be built that describes them. This is achieved by building an xAI model on these cases and extracting the qualitative model from its explanation. A more sophisticated approach for evaluation is to build an explainable difference model on top of the potentially critical cases and the predictions of the original black-box model. This model highlights and explains the uncertainties of the black-box model and the unpredicted critical scenarios. This explanation is used to improve the embedded ML model, improving dependability by fault prevention and tolerance.

In my experiment, I trained a DT with these critical samples and used this to extract a qualitative model from it.

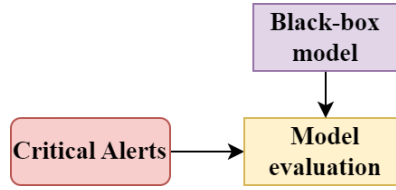


Figure 5.16: Model Evaluation

The last step is *model improvement*. (Figure 5.17) This can be done on a prediction level or a model level. Prediction-level improvement means that we compare the prediction of the model with the descriptive model of the unpredicted critical cases. In my experiment if a sample was predicted negative and it also matched a critical case according to the qualitative model it automatically got a positive label. This way the number of missed alerts decreased by 60%. The model-level improvement consists of modifying the cost function using the model describing the unpredicted critical cases to train a model that is precise at detecting critical scenarios. Future work involves trying out model-level improvement in practice.

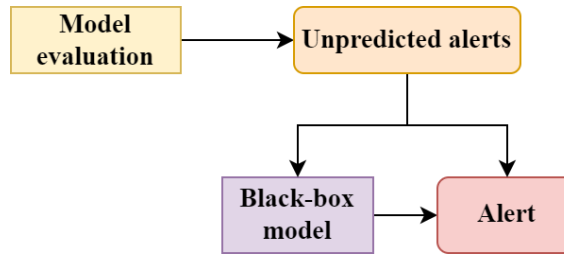


Figure 5.17: Model Improvement

Chapter 6

Conclusion and Future work

This report has tried to provide an overview of the technologies used in this research, which are qualitative modeling, explainable AI, and EPA. It has shown how qualitative models are built and how they can be used as an abstract representation of a system model, therefore allowing efficient qualitative reasoning and the usage of efficient discrete formal methods for system verification. It has shown state-of-the-art methods for model interpretability by using explainable models or generating explanations for different aspects of black-box models by xAI tools therefore providing transparent model validation. It has also presented the fundamental concepts of dependability, how can the dependability of a CPS be ensured and what type of challenges have to be dealt with when operating an embedded model in a critical environment where there are risks involved. It has tried to familiarize the concept of EPA which is a method for evaluating how local faults cause system-level consequences. It has outlined the challenges and methods of impact analysis of embedded ML models. This report focused on providing an efficient method for analyzing the faults of an ML model in the system it is embedded in and its possible consequences. Moreover, it has also shown how to examine the possible failures a model can cause and improve it to perform better considering these failures.

In conclusion, this research has provided a universal and explainable approach for the impact analysis of embedded AI models to improve system dependability by integrating xAI into EPA. This method provides a comprehensive hazard analysis, which helps identify critical failures and by projecting them back onto the embedded model, it pinpoints its weaknesses.

In future work, the model-level improvement would be tried out in practice, this means experimenting with building custom cost functions for models based on the critical scenarios in the system identified by the impact analysis. Another unaddressed challenge is the requirement of absolute sensitivity (in an ideal case, no false negatives, thus escaping faults) and simultaneously of high specificity (low rate of false positives, thus few false alarms) of the surrogate model. This was put in the background in the current research and substituted with a best-effort approach, as they do not influence the core algorithm. Another candidate to be evaluated is Inselberg's nested cavities method [22]. This method can guarantee (at least for the teaching set) a perfect sensitivity and gradually improve the specificity by model refinement (the algorithm generates a converging series of upper and lower approximations guided by a Reed-Mueller normal form styled logic instead of the usual AND-OR). In this research, the assumption was that an unpredicted failure causes more damage than a false alarm, yet this question has to be also examined in the future.

Bibliography

- [1] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques, 2019. URL <https://arxiv.org/abs/1909.03012>.
- [2] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. *Department of Computing Science Technical Report Series*, 2001.
- [3] Hubert Baniecki, Wojciech Kretowicz, Piotr Piatyszek, Jakub Wisniewski, and Przemysław Biecek. dalex: Responsible machine learning with interactive explainability and fairness in python. *Journal of Machine Learning Research*, 22(214):1–7, 2021. URL <http://jmlr.org/papers/v22/20-1473.html>.
- [4] Radim Belohlavek. Introduction to formal concept analysis. *Palacky University, Department of Computer Science, Olomouc*, 47, 2008.
- [5] Przemysław Biecek. Dalex: Explainers for complex predictive models in r. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL <http://jmlr.org/papers/v19/18-416.html>.
- [6] Przemysław Biecek and Tomasz Burzykowski. *Explanatory Model Analysis*. Chapman and Hall/CRC, New York, 2021. ISBN 9780367135591. URL <https://pbiecek.github.io/ema/>.
- [7] Andrea Bondavalli and Francesco Brancati. *Certifications of Critical Systems-The CECRIS Experience*. River Publishers, 2017.
- [8] Yizong Cheng and George M Church. Biclustering of expression data. In *Ismb*, volume 8, pages 93–103, 2000.
- [9] William Cohen. Fast effective rule induction. *Twelfth International Conference on Machine Learning: 1995*, 95, 10 2000. DOI: 10.1016/B978-1-55860-377-6.50023-2.
- [10] P. Cousot. Abstract interpretation based formal methods and future challenges, invited paper. In R. Wilhelm, editor, « *Informatics — 10 Years Back, 10 Years Ahead* », volume 2000 of *Lecture Notes in Computer Science*, pages 138–156. Springer-Verlag, 2001.
- [11] Sanjeeb Dash, Oktay Günlük, and Dennis Wei. Boolean decision rules via column generation, 2020.

- [12] Veloso Bruno Ribeiro Rita Davari, Narjes and Joao Gama. MetroPT-3 Dataset. UCI Machine Learning Repository, 2023. DOI: <https://doi.org/10.24432/C5VW3R>.
- [13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. DOI: <https://doi.org/10.1111/j.2517-6161.1977.tb01600.x>. URL <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1977.tb01600.x>.
- [14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.
- [15] András Földvári and András Pataricza. Handling uncertainty in error propagation analysis. In *Proceedings of the 30th Minisymposium*, page 29–32. Budapest University of Technology and Economics, 2023. DOI: 10.3311/minisy2023-008. URL <http://dx.doi.org/10.3311/minisy2023-008>.
- [16] Kenneth D. Forbus. Chapter 9 qualitative modeling. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 361–393. Elsevier, 2008. DOI: [https://doi.org/10.1016/S1574-6526\(07\)03009-X](https://doi.org/10.1016/S1574-6526(07)03009-X). URL <https://www.sciencedirect.com/science/article/pii/S157465260703009X>.
- [17] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000. DOI: <https://doi.org/10.1006/jcss.1997.1504>. URL <https://www.sciencedirect.com/science/article/pii/S002200009791504X>.
- [18] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision making and a “right to explanation”. *AI Magazine*, 38(3):50–57, September 2017. ISSN 2371-9621. DOI: 10.1609/aimag.v38i3.2741. URL <http://dx.doi.org/10.1609/aimag.v38i3.2741>.
- [19] David Gunning, Eric Vorm, Jennifer Yunyan Wang, and Matt Turek. Darpa’s explainable ai (xai) program: A retrospective. *Applied AI Letters*, 2(4):e61, 2021. DOI: <https://doi.org/10.1002/ail2.61>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/ail2.61>.
- [20] Karthik S. Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights, 2019. URL <https://arxiv.org/abs/1707.01212>.
- [21] Swagatam Haldar, Diptikalyan Saha, Dennis Wei, Rahul Nair, and Elizabeth M. Daly. Interpretable differencing of machine learning models, 2023. URL <https://arxiv.org/abs/2306.06473>.
- [22] Alfred Inselberg. *Parallel Coordinates: Visualization, Exploration and Classification of High-Dimensional Data*, pages 643–680. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-33037-0. DOI: 10.1007/978-3-540-33037-0_25. URL https://doi.org/10.1007/978-3-540-33037-0_25.

- [23] Intel. Intel/intel-xai-tools: Explainable ai tooling (xai). xai is used to discover and explain a model’s prediction in a way that is interpretable to the user. relevant information in the dataset, feature-set, and model’s algorithms are exposed. URL <https://github.com/intel/intel-xai-tools>.
- [24] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., USA, 1988. ISBN 013022278X.
- [25] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_425. URL https://doi.org/10.1007/978-0-387-30164-8_425.
- [26] Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction To Cluster Analysis*, chapter 5. 01 1990. ISBN 0-471-87876-6. DOI: 10.2307/2532178.
- [27] Imre Kocsis. Qualitative models in resilience assurance. Ph.D. dissertation. 2019.
- [28] Bartłomiej Kubica. *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization and Similar Problems: From Inequalities Systems to Game Solutions*. 01 2019. ISBN 978-3-030-13794-6. DOI: 10.1007/978-3-030-13795-3.
- [29] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations, 2018. URL <https://arxiv.org/abs/1711.00848>.
- [30] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.
- [31] Paul D. McNicholas. Model-based clustering. *Journal of Classification*, 33(3):331–373, Oct 2016. ISSN 1432-1343. DOI: 10.1007/s00357-016-9211-9. URL <https://doi.org/10.1007/s00357-016-9211-9>.
- [32] Mary M. Moya and Don R. Hush. Network constraints and multi-objective optimization for one-class classification. *Neural Networks*, 9(3):463–474, 1996. ISSN 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(95\)00120-4](https://doi.org/10.1016/0893-6080(95)00120-4). URL <https://www.sciencedirect.com/science/article/pii/0893608095001204>.
- [33] András Pataricza. Model-Based Dependability Analysis. DSc thesis. 2008.
- [34] Zdzislaw Pawlak, Lech Polkowski, and Andrzej Skowron. Rough set theory. *KI*, 15(3):38–39, 2001.
- [35] D. Pearce and E. Grant. A qualitative modelling environment for design & diagnosis of automation. In *Proceedings of the 2nd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - Volume 1*, IEA/AIE ’89, page 192–196, New York, NY, USA, 1989. Association for Computing Machinery. ISBN 0897913205. DOI: 10.1145/66617.66642. URL <https://doi.org/10.1145/66617.66642>.
- [36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

- [37] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.
- [38] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016. URL <https://arxiv.org/abs/1602.04938>.
- [39] Bernd Steinbach and Christian Posthoff. *Boolean Differential Calculus*, pages 225–295. Springer International Publishing, Cham, 2022. ISBN 978-3-030-88945-6. DOI: 10.1007/978-3-030-88945-6_6. URL https://doi.org/10.1007/978-3-030-88945-6_6.
- [40] Qinghua Zhang, Qin Xie, and Guoyin Wang. A survey on rough set theory and its applications. *CAAI Transactions on Intelligence Technology*, 1(4):323–333, 2016. ISSN 2468-2322. DOI: <https://doi.org/10.1016/j.trit.2016.11.001>. URL <https://www.sciencedirect.com/science/article/pii/S2468232216300786>.

Appendix

A.1 Abbreviations

AI Artificial Intelligence

AGNES Agglomerative Nesting

AIX360 IBM AI Explainability 360

BRCG Boolean Rules via Column Generation

CG Column Generation

CNF Conjunctive Normal Form

CPS Cyber-Physical Systems

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DT Decision Tree

EM Expectation-Maximization

EPA Error Propagation Analysis

FN False Negative

FP False Positive

IMD Interpretable Model Differencing

IT Information Technology

JST Joint Surrogate Tree

LP Linear Programming

MDL Minimum Description Length

MIP Master Integer Program

ML Machine Learning

NN Neural Network

OT Operational Technology

PN Predicted Negative

PP Predicted Positive

QM Qualitative Modeling

RDML Risk-Driven Machine Learning

RF Random Forest

RIPPER Repeated Incremental Pruning to Produce Error Reduction

ROC Receiver Operating Characteristic

SAT Satisfiability

SMOTE Synthetic Minority Oversampling Technique

TN True Negative

TP True Positive

xAI Explainable Artificial Intelligence

A.2 Syntax of the rulesets

The syntax of the rulesets provided by the xAI models presented in this research report is the same. Representing the grammar of these rulesets with *BNF notation* looks like the following:

```
rule ::= "if" predicate "then" result
predicate ::= '(' predicate ')' | predicate 'V' predicate |
| predicate '^' predicate | condition
condition ::= '[' feature operator value ']'
operator ::= "==" | "!=" | "<" | "<=" | ">" | ">="
feature ::= identifier
value ::= number
result ::= number | identifier
```

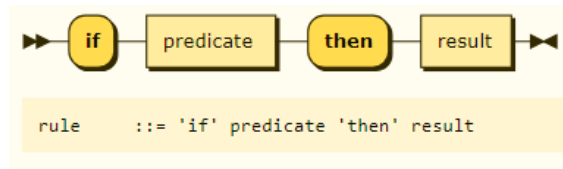


Figure A.2.1: Rule syntax

A rule begins with the keyword 'if', followed by a *predicate*, then a 'then' keyword, and finally the *result* (Figure A.2.1). A predicate can be between parentheses, it can be a *conjunction of predicates*, a *disjunction of predicates*, or it can be a *condition* (Figure A.2.2).

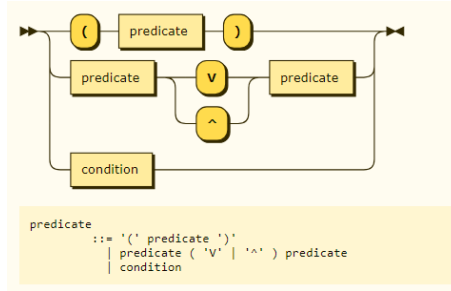


Figure A.2.2: Predicate syntax

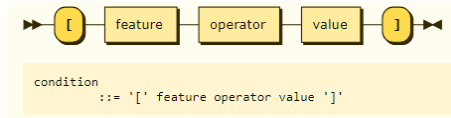


Figure A.2.3: Condition syntax

A condition contains a *feature* between square brackets, an *operator*, and a *value*. (Figure A.2.3) The name of a feature is an *identifier*, and the value is a *number*. An operator can be ==, !=, <, <=, >, >=, as shown in Figure A.2.4.

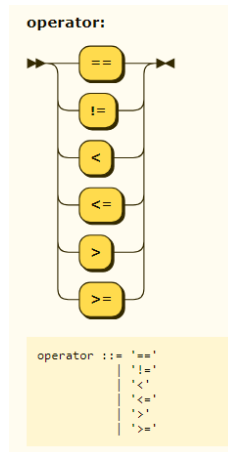


Figure A.2.4: Operator syntax

A.3 Experiments using the Adult Dataset

The Adult dataset ¹, also known as the "Census Income" dataset, is a popular dataset in machine learning for binary classification tasks. It contains 48,842 instances and 14 attributes, including age, education level, occupation, hours worked per week, marital status, and race. The goal is to predict whether a person's annual income is above or below \$50,000 based on these attributes.

¹<https://archive.ics.uci.edu/dataset/2/adult>

A.3.1 Adding semantic redundancy

This experiment examines how the structure of a decision tree (DT) changes when semantic redundancy is added to the training data. In this example, four new columns are added to the Adult dataset: *education-level*, *occupation-type*, *relationship-type*, and *continent*.

The *education-level* feature is an abstraction of the *education* feature, with its 6 possible categorical values abstracted into {basic, middle, acad}. The *occupation-type* feature is an abstraction of the *occupation* feature, reducing the 14 possible values to 9 values based on the ISCO-08 standard².

The *relationship-type* feature is created by abstracting the 6 values of the *relationship* feature into {single, married}. The *continent* feature is created by abstracting the native countries of individuals into {Developed, Europe, Third World, USA}.

As a first step, a decision tree was trained using the original dataset with a depth limit of 4 (shown in Figure A.3.1). This produced the following results:

Accuracy: 0.8487640104406572
Balanced accuracy: 0.7466407268940976
Precision: 0.7570175438596491
Recall: 0.5493316359007002

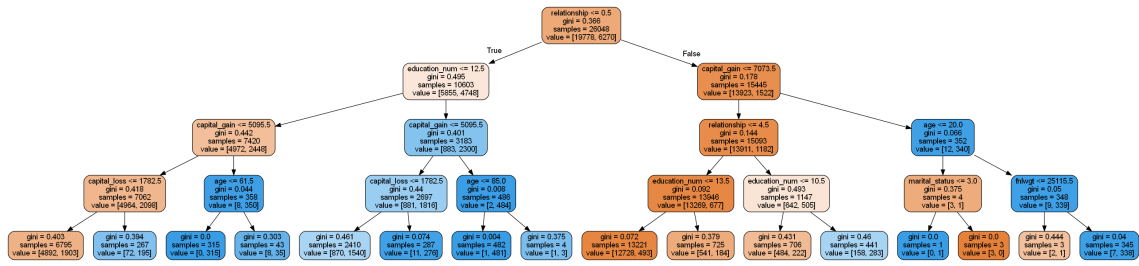


Figure A.3.1: DT adult dataset

After adding the semantic redundancy, another DT was trained using the newly created dataset (shown in Figure A.3.2) with the following results:

Accuracy: 0.8318407960199005
Balanced accuracy: 0.7265835623973481
Precision: 0.7422615287428932
Recall: 0.513548951048951

It can be seen, that the performance metrics did not improve, but the second DT did use the features: */emphrelationship-type* and *occupation-type*.

A.3.2 Explainable difference model

In this experiment, a DT model and a BRCG model were trained on the Adult dataset and then they were compared using the IMD algorithm to build a difference model.

²International Standard Classification of Occupations: https://en.wikipedia.org/wiki/International_Standard_Classification_of_Occupations

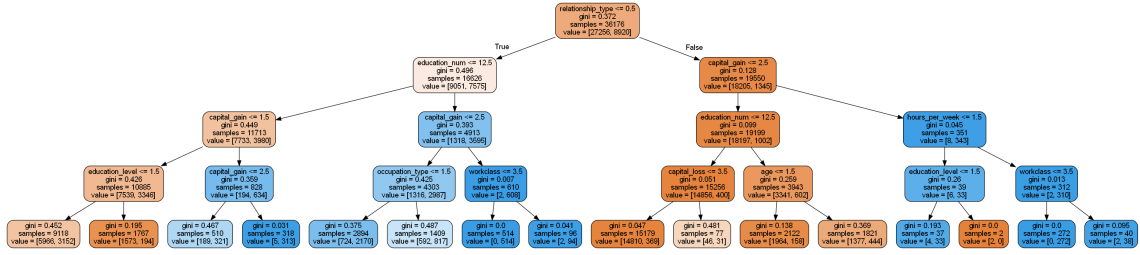


Figure A.3.2: DT with semantic redundancy

As a first step, the dataset is loaded into a pandas dataframe, so that the models can be trained with it. The categorical values have to be encoded, for this the LabelEncoder was used.

```
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df = df.drop(df_categorical.columns,axis=1)
df = pd.concat([df,df_categorical],axis=1)
```

As a next step, the train and test set is split:

```
train, test = train_test_split(df, test_size=0.2, random_state=42)
y_train = train[TARGET_COLUMN]
x_train = train.drop(columns=[TARGET_COLUMN])
y_test = test[TARGET_COLUMN]
x_test = test.drop(columns=[TARGET_COLUMN])
```

A.3.2.1 Training the DT model

```
d_tree_clf = tree.DecisionTreeClassifier(max_depth=4)
d_tree_clf = d_tree_clf.fit(x_train, y_train)
```

After training the model was evaluated. The results are:

```
Accuracy: 0.8487640104406572
Balanced accuracy: 0.7466407268940976
Precision: 0.7570175438596491
Recall: 0.5493316359007002
```

A.3.2.2 Training the BRCG

For using the BRCG algorithm, the features have to be binarized:

```
fb = FeatureBinarizer(negations=True)
X_train_fb = fb.fit_transform(x_train)
x_test_fb = fb.transform(x_test)
```

Then the model can be trained:

```
brcg_explainer = BRCG(silent=True)
brcg_explainer.fit(X_train_fb, y_train)
```

After a training time of 45.6 sec, the performance metrics values are the following:

```
Accuracy: 0.8189774297558728
Balanced accuracy: 0.6870675649114708
Precision: 0.7028985507246377
Recall: 0.4322087842138765
```

The **ruleset the BRCG model** learned is the following:

```
if
([education_num > 11.0] ^ [relationship <= 0.0]) v
([education_num > 11.0] ^ [relationship > 4])
then
1
```

A.3.2.3 Creating the difference model

At first, the predictions of both models have to be evaluated for the training set. The evaluation shows that the two sets of predictions differ by 5%. Then, the IMD model is fit with a depth limit of 6.

```
imd = IMDExplainer()
imd.fit(x_train, y1, y2, max_depth=6)
```

The ruleset of the model looks like the following:

```
[IF capital_loss <= 1820.5000 AND
education_num > 10.5000 AND
education_num <= 11.5000 AND
relationship > 4.5000 AND
capital_gain <= 5119.0000
THEN PREDICT class = 1,
IF capital_loss > 1820.5000 AND
education_num <= 11.5000 AND
relationship <= 0.5000 AND
capital_gain <= 5119.0000
THEN PREDICT class = 1,
IF capital_loss > 1820.5000 AND
education_num > 10.5000 AND
education_num <= 11.5000 AND
relationship > 4.0000 AND
capital_gain <= 5119.0000
THEN PREDICT class = 1,
IF education_num <= 11.5000 AND
relationship <= 0.5000 AND
capital_gain > 5119.0000 AND
```

```

    capital_gain <= 7073.5000
THEN PREDICT class = 1,
IF age <= 20.0000 AND
    education_num <= 11.5000 AND
    relationship <= 2.5000 AND
    capital_gain > 7073.5000
THEN PREDICT class = 1,
IF fnlwgt <= 24872.5000 AND
    age > 20.0000 AND
    relationship <= 0.5000 AND
    education_num <= 11.5000 AND
    capital_gain > 7073.5000
THEN PREDICT class = 1,
IF fnlwgt > 24872.5000 AND
    age > 20.0000 AND
    education_num <= 11.5000 AND
    capital_gain > 7073.5000
THEN PREDICT class = 1,
IF education <= 8.0000 AND
    capital_loss <= 1794.0000 AND
    relationship <= 0.5000 AND
    education_num > 11.5000 AND
    capital_gain <= 4782.0000
THEN PREDICT class = 1,
IF education_num > 11.5000 AND
    relationship > 0.5000 AND
    relationship <= 4.5000 AND
    capital_gain > 7073.5000
THEN PREDICT class = 1]

```

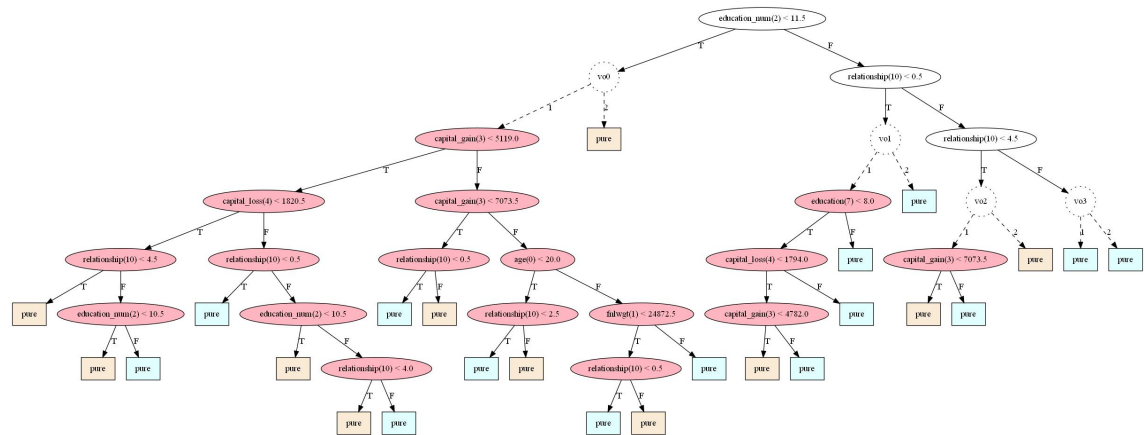


Figure A.3.3: JST of Adult dataset

The obtained JST (in Figure A.3.3) shows, that the DT has a much more complicated decision process than the BRCG, which is also confirmed by the fact that the BRCG's ruleset is very simple compared to the structure of the DT. The pink nodes correspond to the DT's decisions where they differ from the common decisions. The presence of orange nodes would represent those decisions where the BRCG decides differently than the DT,

but this is not the case. It also can be seen that the two models generalize the same way, and the DT instance is more refined compared to the BRCC instance.

A.4 Experiments using the APS Failure at Scania Trucks dataset

The APS Failure at Scania Trucks dataset¹, also known as the Air Pressure System (APS) Failure dataset, is designed for predictive maintenance tasks in the automotive industry which is a critical system's context. This dataset was created by Scania, a major Swedish manufacturer of commercial vehicles, specifically trucks and buses. It consists of sensor data collected from the air pressure system (APS) of Scania trucks and is intended to facilitate the development of models that can predict potential failures in the APS, a critical component for vehicle safety. It is a highly imbalanced dataset with an imbalance ratio of 1 : 100. It has 171 features representing various sensor readings and signals. These features are anonymized because of proprietary reasons. That is why explanations of models can hardly be interpreted when using this dataset, because of the lack of domain knowledge.

A.4.1 High level of imbalance

The first interesting result when using this dataset was that as it has a high level of imbalance, the models tended to overgeneralize. A prime example of this is that as the BRCC algorithm tries to come up with a relatively simple ruleset with a relatively high coverage of samples it produces the following ruleset:

```
if false then 1
```

This means every sample is automatically predicted a negative label. In theory, if a dataset with an imbalance ratio of 1 : 100, meaning that if it has 100 elements, it contains 1 positive sample, so with this ruleset, a 99% accuracy can be achieved. In practice, on the test set it achieved a 91% accuracy, as its performance metrics were the following:

```
Accuracy: 0.9147787648233944  
Balanced accuracy: 0.5  
Precision: 0.0  
Recall: 0.0
```

The precision and recall values of 0 confirm the fact, that the model did not make any positive prediction.

A.4.2 Difference model on the positive samples

As the level of imbalance is very high as presented above, building a difference model over the entire dataset would result in a very small number of differences which is difficult to analyze. That is why a new approach was tried out that manifests in building a difference model only over the originally positive samples.

¹<https://archive.ics.uci.edu/dataset/421/aps+failure+at+scania+trucks>

In this experiment, a random forest (RF) and a neural network (NN) model were trained on the APS dataset and their predictions were exported into CSV files. Then the training data was built for the IMD model.

```
# Concatenating the features, labels, predictions in a single dataframe
df = pd.concat([df_train, df_y_train, df_y_train_nn, df_y_train_rf], axis=1)

# Selecting the originally positive samples
df_err = df[(df.y_original == 1)]

# Separating the features, labels, predictions
df_err_y_train = df_err["y_original"]
df_err_y_nn = df_err["y_nn"]
df_err_y_rf = df_err["y_rf"]
df_err_x_train = df_err.drop(["y_original", "y_nn", "y_rf"], axis=1)
```

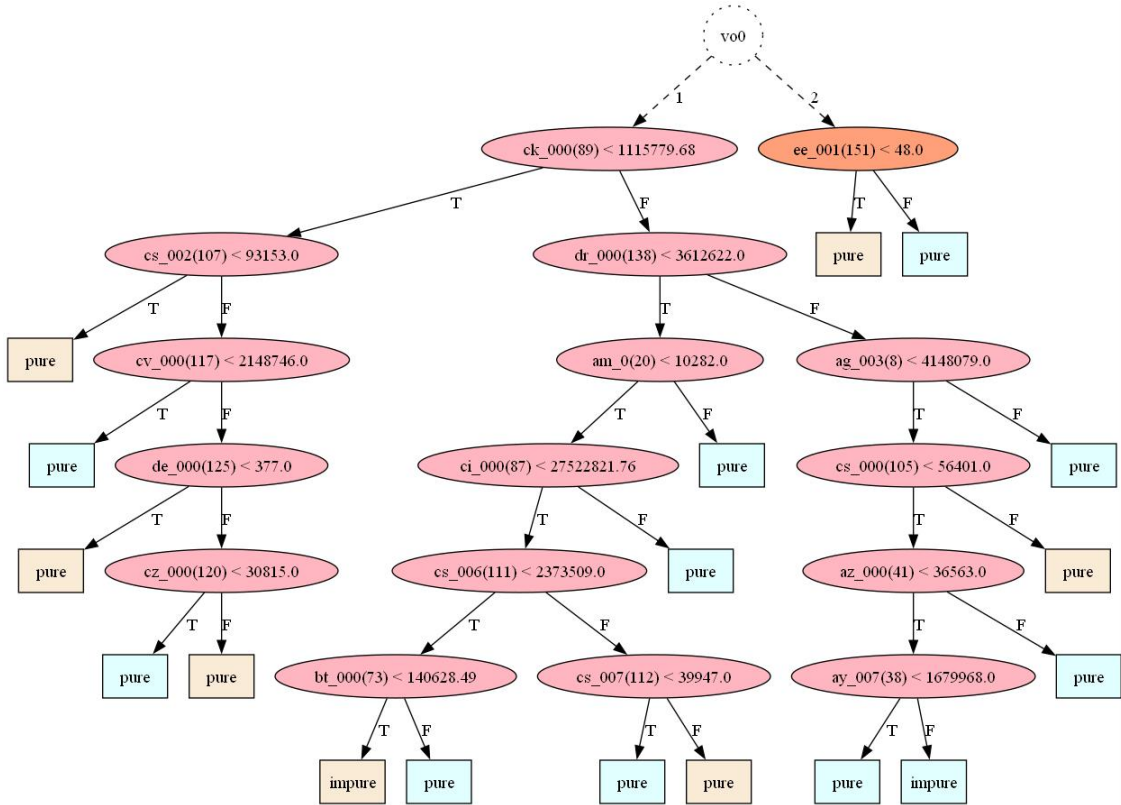


Figure A.4.1: JST over positive samples APS dataset

After training an IMD instance on this data, we get the JST shown in Figure A.4.1. This shows that the NN is a much more refined model compared to the RF, examining a lot of features when making predictions on the boundary region, whereas the RF only decides based on one split condition of a feature.

A.5 Experiments using the MetroPT-3 dataset

This dataset contains, from a metro train in an operational context, readings from pressure, temperature, motor current, and air intake valves collected from a compressor's Air

Production Unit (APU). This dataset reveals real predictive maintenance challenges encountered in the industry. It can be used for failure predictions, anomaly explanations, and other tasks.

The dataset was collected to support the development of predictive maintenance, anomaly detection, and remaining useful life (RUL) prediction models for compressors using deep learning and machine learning methods. It consists of multivariate time series data obtained from several analog and digital sensors installed on the compressor of a train. The data span between February and August 2020 and includes 15 signals, such as pressures, motor current, oil temperature, and electrical signals of air intake valves. The monitoring and logging of industrial equipment events, such as temporal behavior and fault events, were obtained from records generated by the sensors. The data were logged at 1Hz by an onboard embedded device. The MetroPT-3 dataset contains 1.5 million unlabeled samples.

A.5.1 Training an advanced model with preprocessed data

A.5.1.1 Preprocessing

The workflow of preprocessing the data for training a neural network model was inspired by a publicly available group project report from the University of Huston ².

At first, the first column, which is unnamed and contains the indices of the samples was removed. Then the samples were labeled according to the failure reports in the dataset description shown in Figure A.5.1.

Nr.	Start Time	End Time	Failure	Severity	Report
#1	4/18/2020 0:00	4/18/2020 23:59	Air leak	High stress	
#1	5/29/2020 23:30	5/30/2020 6:00	Air Leak	High stress	Maintenance on 30Apr at 12:00
#3	6/5/2020 10:00	6/7/2020 14:30	Air Leak	High stress	Maintenance on 8Jun at 16:00
#4	7/15/2020 14:30	7/15/2020 19:00	Air Leak	High stress	Maintenance on 16Jul at 00:00

Figure A.5.1: Failure reports

Every sample in the time intervals of failures got a positive label. This way the dataset had an imbalance ratio of 2 : 100.

As the next step, the negative samples are subsampled to match the number of positive samples.

As a final step, the outliers were found and dropped from the dataset. The outliers were found the following way. Let $Q1$ be the 25th percentile and $Q3$ be the 75th percentile. Then let $IQR = Q3 - Q1$ denote the interquartile range. A value is an outlier if is less than $Q1 - 1.5 * IQR$ or greater than $Q3 + 1.5 * IQR$.

²<https://github.com/harveyphm/MetroPT-3-Anomaly-Detection>

A.5.1.2 Training a neural network with preprocessed data

As a highly advanced model, a neural network was used as a failure prediction model trained on the MetroPT-3 dataset. This neural net consists of two hidden layers having 64 neurons and the activation function of *relu* and an output layer of a single neuron using the *sigmoid* activation function. The configuration of the neural network is shown in Figure A.5.2.

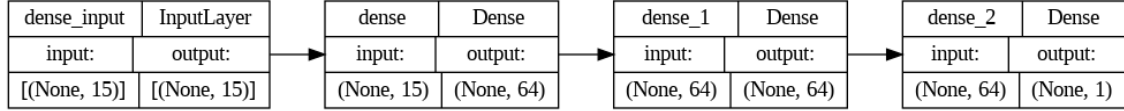


Figure A.5.2: NN configuration

The model creation looks like the following:

```
# Model creation
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu')) # 1st hidden
model.add(Dense(64, activation='relu')) # 2nd hidden
model.add(Dense(1, activation='sigmoid')) # output
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
```

Standardization of numerical values before training:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

As a first step, I trained a model on the preprocessed dataset, and got the results shown in figure A.5.3. Then I trained the model without data preprocessing, but I still standardized the numerical values for the NN using the StandardScaler (Figure A.5.4). As a last step, I trained the same model with raw labeled data without preprocessing and standardization (Figure A.5.5).

From the results, it can be seen that the one trained without preprocessing performed slightly worse than the one trained with preprocessing. The relatively high performance for data without preprocessing is due to the fact that as neural networks are highly advanced models they are also capable of learning unbalanced datasets with relatively high accuracy. The results also show that the model trained on unstandardized data performed significantly worse compared to the other ones.

A.5.2 Difference model on positive samples

For the difference model, I used the neural network model described above and a random forest model, trained on the exact same data. The difference models for this dataset were created using the exact same procedure as described in section A.4.2.

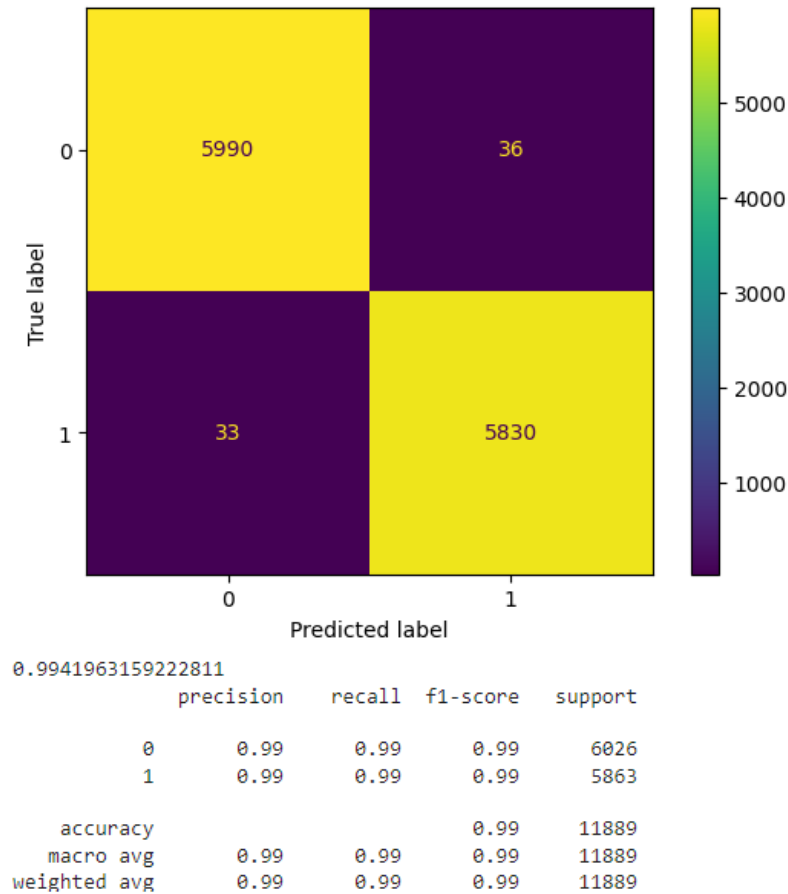


Figure A.5.3: NN results with preprocessed data

A.5.2.1 IMD difference model

The first type of difference model I trained was an IMD. The resulting JST is shown in Figure A.5.6

This is a relatively balanced JST, showing the common decisions of the two models and the points where they decide based on different factors.

A.5.2.2 XOR difference model

Another type of difference model I created is that I applied the XOR function to the predictions of the two models so that the samples where the two models contradicted got a positive label. This set of labels was created using the following code:

```
y_diff = np.bitwise_xor(df_err_y_rf, df_err_y_nn)
```

Here, the data frame `df_err_y_rf` and `df_err_y_nn` contain the predictions of the RF and NN for all of the positive samples of the dataset.

Then, I trained a DT with a depth limit of 4 using this set of labels and the features from the dataset. The resulting DT is shown in figure A.5.7

```
d_tree_clf = tree.DecisionTreeClassifier(max_depth=4)
d_tree_clf = d_tree_clf.fit(df_err_x_train, y_diff)
```

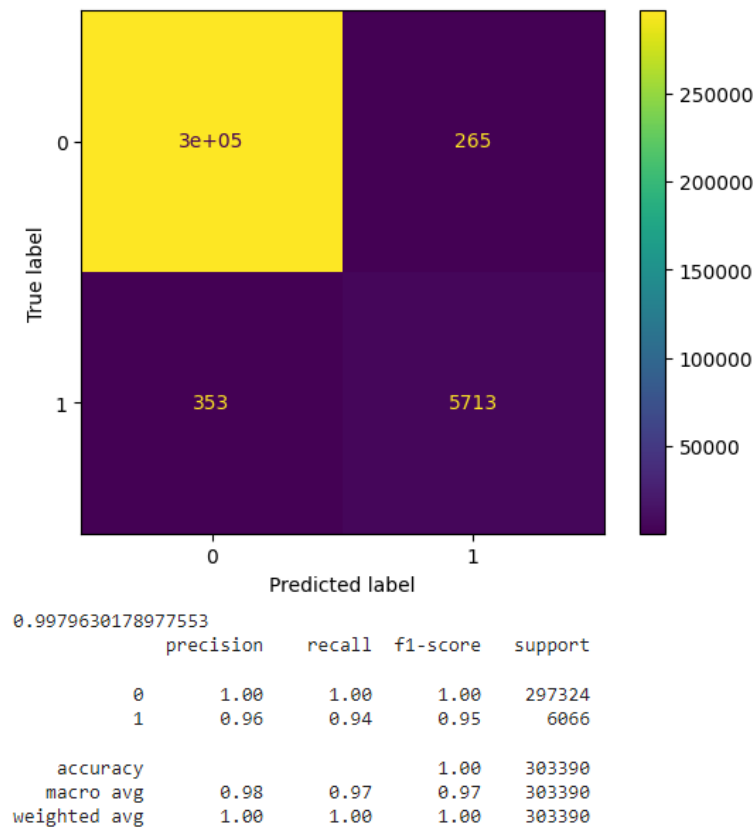


Figure A.5.4: NN results with only standardized data

Here, the `df_err_x_train` variable contains the features of all positive samples.

A.5.2.3 From DT difference model to qualitative model

The following example shows how to convert a decision tree into a qualitative model.

As a first step, I extracted the qualitative landmarks from the decision tree for all features involved in split conditions using the following code:

```
from sklearn.tree import _tree
def get_landmarks(tree, feature_names):
    tree_ = tree.tree_
    feature_name = [ # feature names in splitting conditions of nodes
        # if the node is a leaf, then the feature name is undefined!
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined!"
        for i in tree_.feature
    ]

    n_of_features = feature_names.shape[0]
    # contains the arrays of qualitative landmarks of the features
    landmarks = np.empty(shape=(n_of_features, 1), dtype=object)

    # initializes with empty arrays for every feature
    for i in range(0, n_of_features):
```

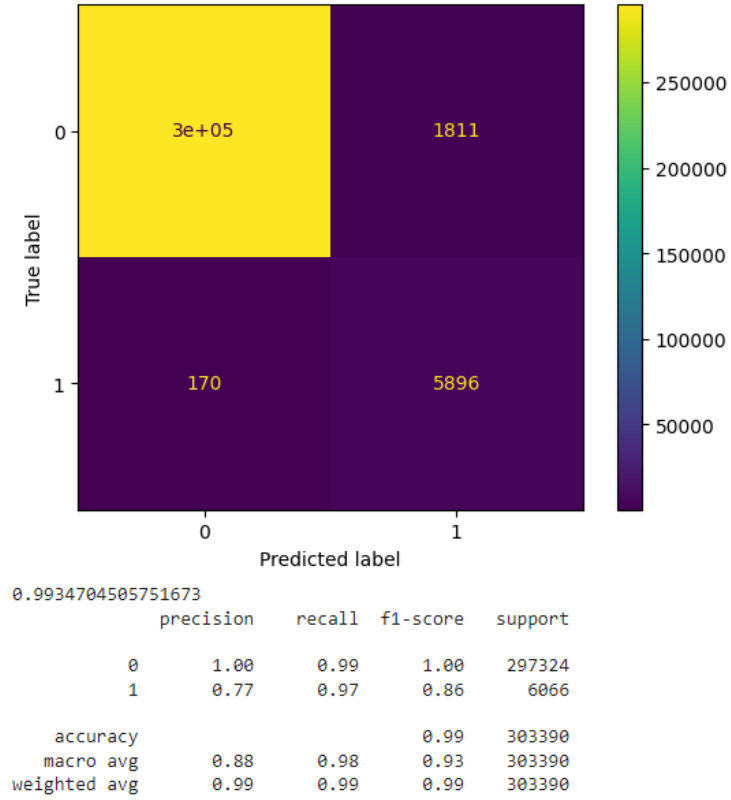


Figure A.5.5: NN results with only labeled data

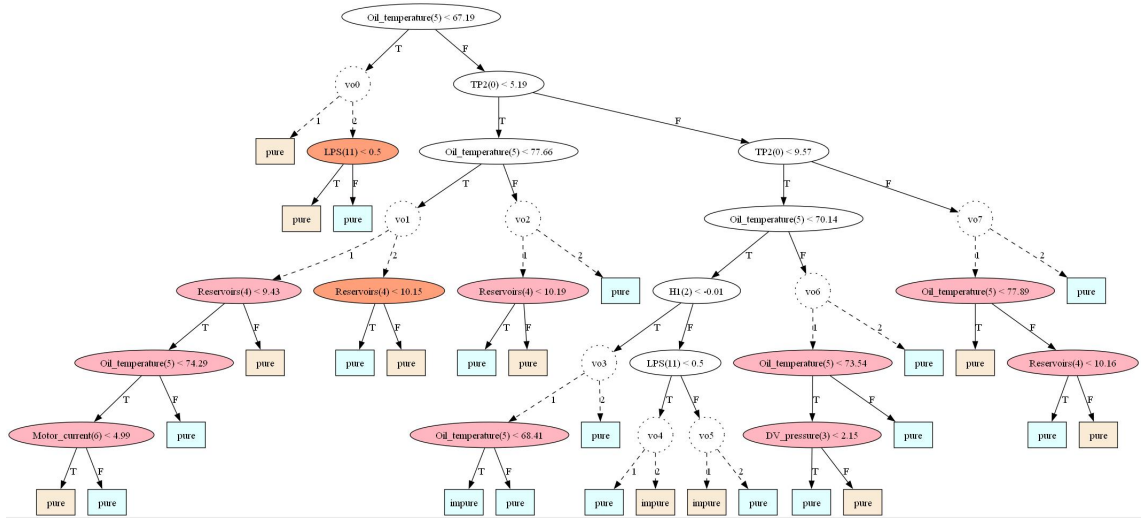


Figure A.5.6: JST of MetroPT-3 dataset

```
landmarks[i][0] = []
```

```
# it traverses the tree and extracts split values as landmarks
def recurse(node, landmarks):
    if tree_.feature[node] != _tree.TREE_UNDEFINED:
        name = feature_name[node]
        threshold = tree_.threshold[node]
        landmarks[tree_.feature[node]][0].append(np.round(threshold, 3))
```

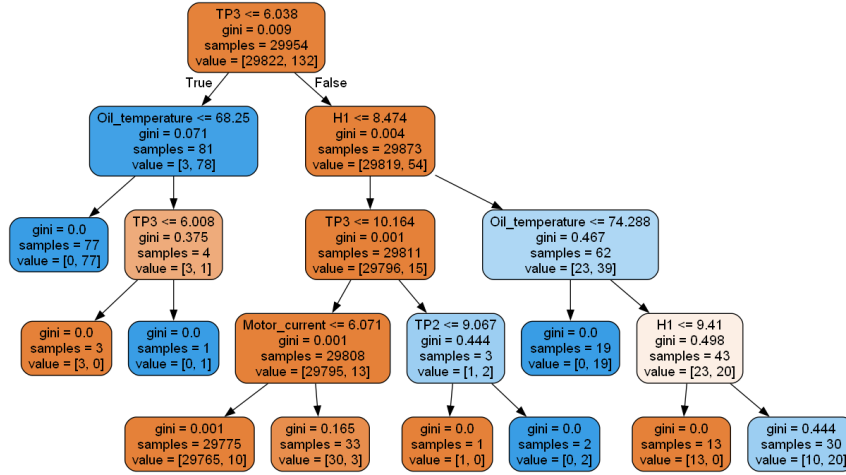


Figure A.5.7: DT XOR difference model

```
landmarks[tree_.feature[node]][0].sort()
recurse(tree_.children_left[node], landmarks)
recurse(tree_.children_right[node], landmarks)
```

```
recurse(0, landmarks)
```

```
return landmarks
```

```
# finally landmarks are extracted by calling the function above
q_landmarks = get_landmarks(d_tree_clf, df_err_x_train.columns)
q_landmarks
```

Running this code using the DT trained above as an input gets the following result:

```
array([[list([])],
       [list([9.427, 10.164])],
       [list([8.474])],
       [list([])],
       [list([6.036])],
       [list([68.25, 74.288])],
       [list([4.967, 6.071])],
       [list([])],
       [list([])],
       [list([0.5])],
       [list([])],
       [list([])],
       [list([])],
       [list([])],
       [list([])]], dtype=object)
```

As the second step using these landmarks the feature domains are partitioned into disjunct intervals. Here I assumed that every feature has $\mathbb{R}$ as its domain. This is achieved with the following code:

```

def create_intervals(landmarks):
    n_of_features = landmarks.shape[0]
    # contains the arrays of the intervals for each feature
    number_lines = np.empty(shape=(n_of_features, 1), dtype=object)

    # initializes with empty arrays for every feature
    for i in range(0, n_of_features):
        number_lines[i][0] = []

    # traverses each landmark and creates intervals from them
    for i in range(0, n_of_features):
        feature = landmarks[i][0]
        if len(feature) > 0:

            # the first interval is (-inf, first_landmark)
            first_landmark = feature[0]
            number_lines[i][0].append(P.closed(-P.inf, first_landmark))

            # each two consecutive landmarks represent
            # the beginning and the end of an interval
            for j in range(1, len(feature)):
                number_lines[i][0].append(P.closed(feature[j-1], feature[j]))

            # the last interval is (last_landmark, inf)
            number_lines[i][0].append(P.closed(feature[-1], P.inf))

    return number_lines

number_lines = create_intervals(q_landmarks)
number_lines

```

For managing intervals the *portion*³ interval logic library was used.
 These intervals extracted from the tree represent discrete qualitative values.

A.5.2.4 Difference model with 8 labels

The last type of difference model I experimented with is one with 8 labels, representing all possible values of the three labels $\{y_{original}, y_{model_1}, y_{model_2}\}$.

The labels for this difference model were created the following way:

```

def create_diff_arr(arr1, arr2, arr3):
    out = np.zeros(len(arr1))
    for i in range(len(arr1)):
        if arr1[i]==0 and arr2[i]==0 and arr3[i]==0:
            out[i]=0
        elif arr1[i]==0 and arr2[i]==0 and arr3[i]==1:
            out[i]=1
        elif arr1[i]==0 and arr2[i]==1 and arr3[i]==0:

```

³<https://pypi.org/project/portion/>

```

        out[i]=2
    elif arr1[i]==0 and arr2[i]==1 and arr3[i]==1:
        out[i]=3
    elif arr1[i]==1 and arr2[i]==0 and arr3[i]==0:
        out[i]=4
    elif arr1[i]==1 and arr2[i]==0 and arr3[i]==1:
        out[i]=5
    elif arr1[i]==1 and arr2[i]==1 and arr3[i]==0:
        out[i]=6
    elif arr1[i]==1 and arr2[i]==1 and arr3[i]==1:
        out[i]=7

return out

```

```
diff_y = create_diff_arr(y.values, df_y_nn["y_nn"].values, df_y_rf["y_rf"].values)
```

Then I trained a DT using these labels on the dataset. This is shown on figure A.5.8

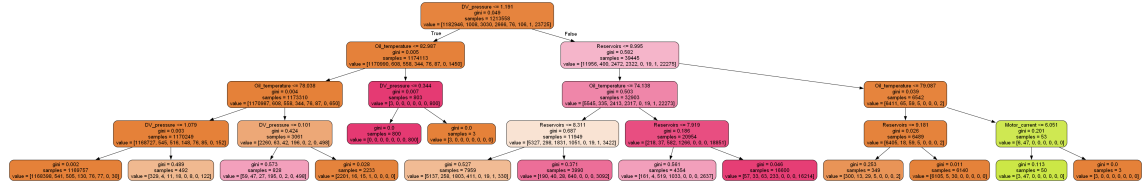


Figure A.5.8: 8 label difference model of MetroPT-3 dataset

A.5.3 Surrogate model creation

This example shows how a *sensitive surrogate model* is built described in section 5.4.4.

As the first method to build a sensitive model is cost-sensitive learning, I trained a DT using class weights of 1/11 for negative samples and 10/11 for positive samples. This code shows how I created the DT.

```

weights = {0:1/11, 1:10/11}
d_tree_clf = tree.DecisionTreeClassifier(max_depth=4, class_weight=weights)
d_tree_clf = d_tree_clf.fit(x_train, y_train)

```

This model produced a confusion matrix shown in Figure A.5.9. This produced a rather high sensitivity of 99%, where on the test set there are only 29 false negative samples out of 6046 samples.

However, I thought we could do better. My method was to iteratively train a DT with cost-sensitive learning, boost the weight of false negative samples, and repeat. I implemented this in the following way:

```

n_of_iter = 10 # number of iterations
boost_ratio = 0.5 # ratio for boosting the sample weights

# initially each sample has a weight of 1
sample_weights = np.ones(shape=(x_train.shape[0]))

```

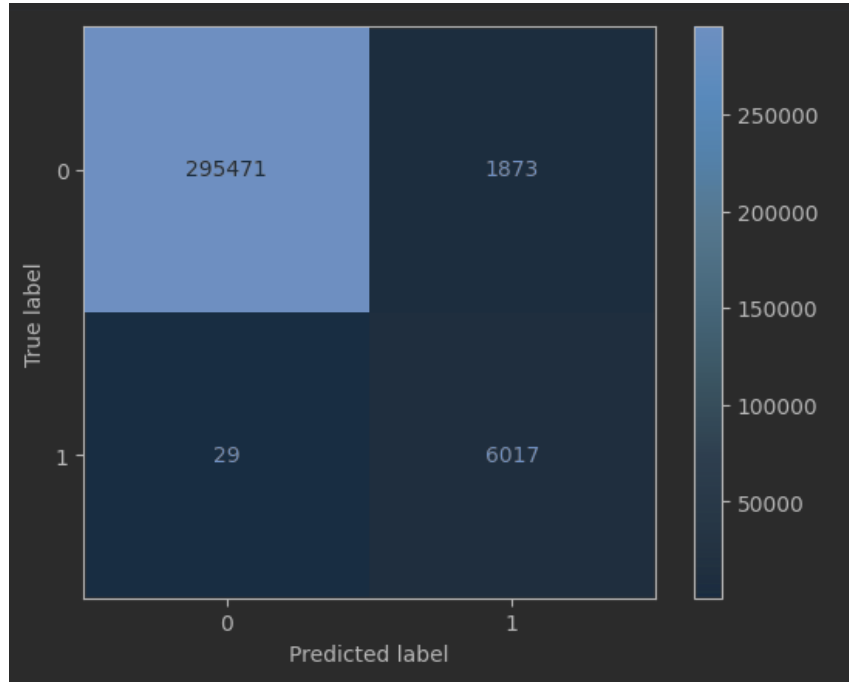


Figure A.5.9: Confusion matrix of DT with cost-sensitive learning

```
for i in range(n_of_iter):
    # training the DT and evaluating on the training set
    d_tree_clf = tree.DecisionTreeClassifier(max_depth=4, class_weight=weights)
    d_tree_clf = d_tree_clf.fit(x_train, y_train, sample_weight=sample_weights)
    y_pred = d_tree_clf.predict(x_train)

    # finding the indices of the FN samples
    y_comp = np.column_stack((y_train, y_pred))
    idx_fn = np.where((y_comp[:, 0] == 1) & (y_comp[:, 1] == 0))[0]

    # boosting their weights
    sample_weights[idx_fn] = sample_weights[idx_fn] * (1+boost_ratio)
```

The resulting model produced the confusion matrix shown in Figure A.5.10. This shows an extraordinarily high sensitivity, covering all positive samples except one. However, the side effect of low specificity also can be seen as the false positive rate increased by 268%.

A.5.4 Evaluation of the surrogate model

In my experiment, I used the boosted DT presented in section A.5.3 as a surrogate model for the NN model presented in section A.5.1.2. After creating the models, I compared them by using the method described in section A.5.2.4. I did not fit a model on it, instead I examined the distribution of the 8 classes.

The results are shown in Table A.1

A.5.5 Critical samples and model improvement

This experiment shows the implementation of the model evaluation workflow in practice.

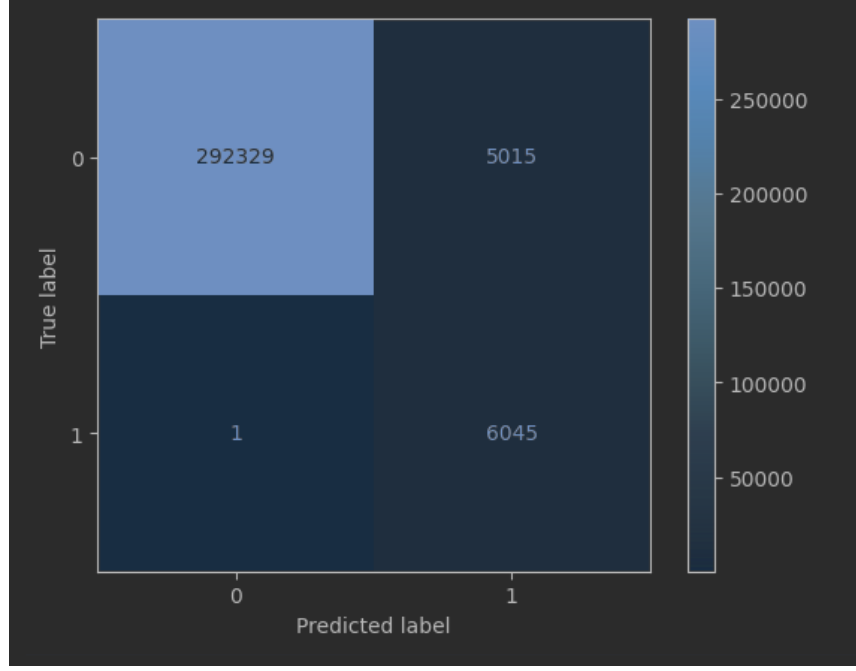


Figure A.5.10: Confusion matrix of boosted DT

label	nn	dt	# of samples	conclusion
0	0	0	292282	both models are good for identifying the majority class
0	0	1	3736	the surrogate model is a more abstract one, more false alarms
0	1	0	47	the production model is a more refined one with few false alarms
0	1	1	1279	the two models generalize similarly
1	0	0	0	the two models cover all samples together
1	0	1	90	surrogate model covers more failures due to abstraction
1	1	0	1	surrogate model covers almost all failures
1	1	1	5955	the two models recognize the majority of faults

Table A.1: Surrogate model evaluation

As a first step, the evaluation of critical cases. After interpreting the dataset description and architecture of the metro's Air Production Unit, a simplified abstract engineering model was built (shown in Figure A.5.11).

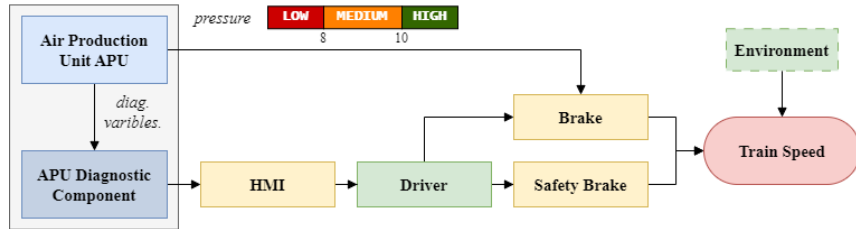


Figure A.5.11: Engineering model MetroPT-3

Based on this model, the APU provides air to the braking system with 3 possible qualitative values: LOW, MEDIUM, HIGH. Based on my interpretation, a MEDIUM or a HIGH pressure value is sufficient for the train to stop safely. However, with a LOW pressure value the stopping of the train can not be guaranteed, so this is a critical value. Projecting this back onto the continuous domain, it can be concluded that a pressure value of less than

8 bars represents a *critical scenario*. As a result, critical scenarios can be described with the condition of $pressure < 8$.

Therefore, potentially critical scenarios can be described by samples that were predicted positive by the surrogate model and they fulfill the condition described above. In my experiment I did exactly this:

```
df_epa_crit = df[(df['y_dt']==1) & (df['Reservoirs']<8)]
```

For the step of model evaluation, I selected those potentially critical samples that were not predicted positive by the advanced model, i.e. the NN model:

```
df_crit = df_epa_crit[(df_epa_crit['y_nn']==0) & \
(df_epa_crit['y_original']==1)]
```

Note, that here I selected only those that originally had a positive label to exclude false positive samples, however in theory it should not contain them anyway. There were 328 of these samples representing undetected critical scenarios out of 1.5 million samples, of which around 30000 are positive.

For model improvement, I created an *improvement dataset*, which means labeling the MetroPT-3 dataset in a way that unpredicted critical cases (obtained in the previous step) get a positive label and the other samples a negative one. As an experiment for prediction-level improvement at every prediction I applied a disjunction between the prediction of the model and the labels of the improvement dataset, as shown in the following code:

```
y_improved = pd.DataFrame({'y_improved': y_nn['y_nn'] | y_ripper['y_ripper']})
```

This means the samples representing undetected critical cases are automatically predicted positive. This resulted in a 60% decrease in false negative values as shown on the two confusion matrices in Figure A.5.12.

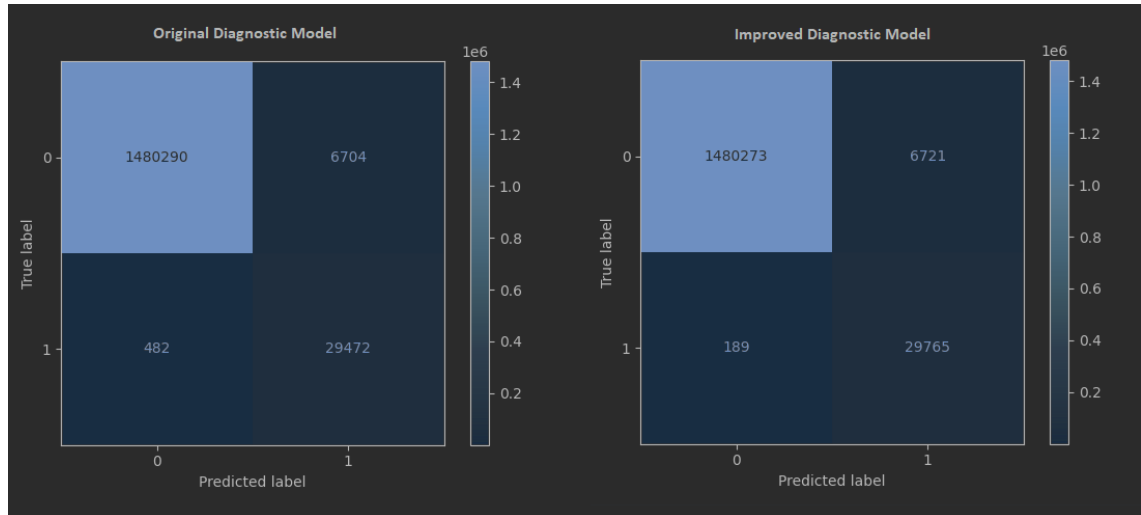


Figure A.5.12: Normal and improved predictions

As another experiment, I tried to fit interpretable-by-design xAI models on this improvement dataset. Note that the imbalance in this is way higher than in the original dataset. In this example, its imbalance ratio is roughly 1 : 4500.

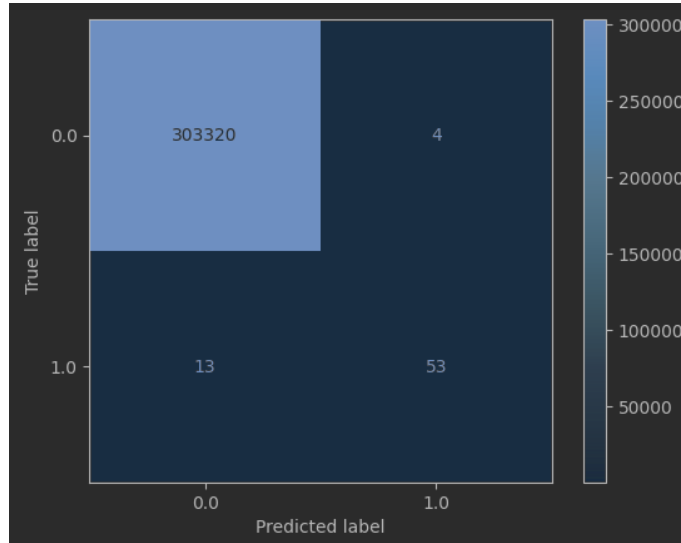


Figure A.5.13: DT Improvement model

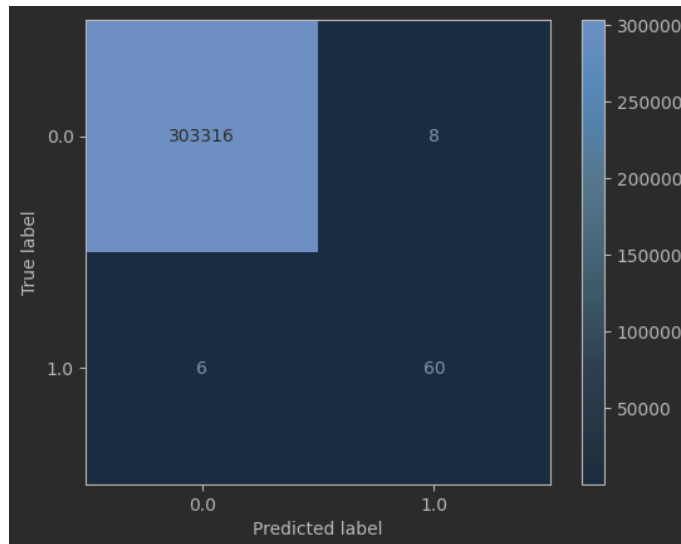


Figure A.5.14: RIPPER Improvement model

I tried fitting DT, RIPPER, and BRCG and then I oversampled the dataset using SMOTE and tried fitting a DT and a BRCG with it. From these models, the results of the RIPPER were the most consistent, which confirms the fact that it is an ML model specially designed for imbalanced datasets. The other result worth noting is that on the imbalanced improvement dataset, the BRCG produced an empty model, which is not surprising when using imbalanced datasets. However, when training on the oversampled one, it produced a single rule as a ruleset, which is the exact same as the rule resulted from EPA, describing the critical cases:

```
if ['Reservoirs <= 7.98'] then predict 1
```

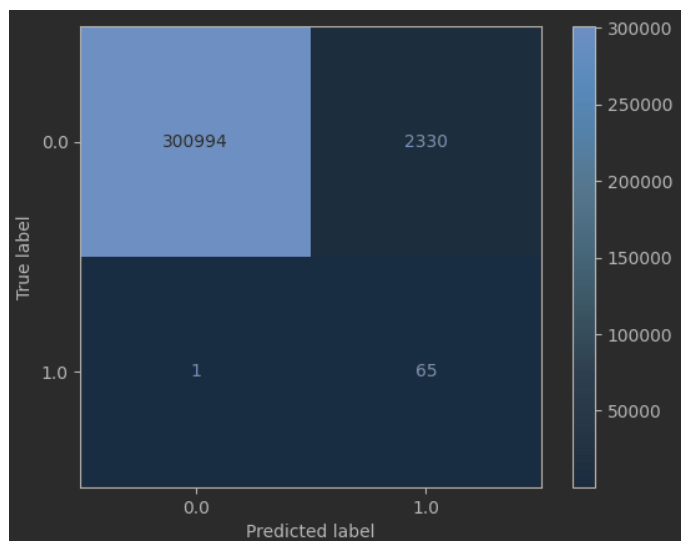


Figure A.5.15: DT oversampled Improvement model

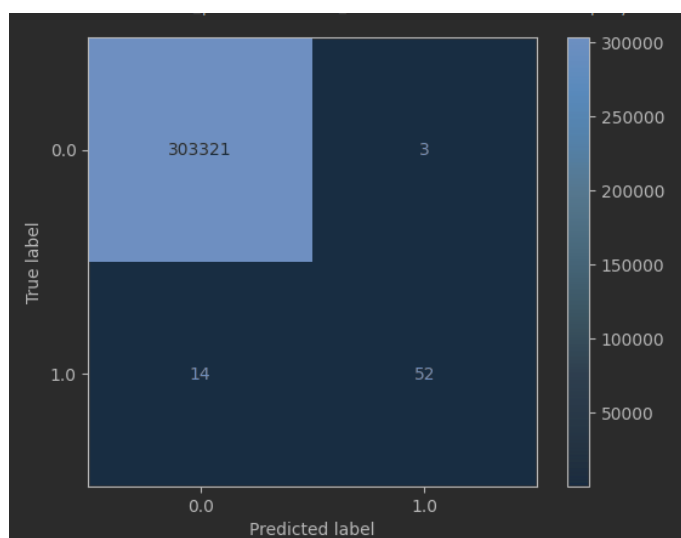


Figure A.5.16: RIPPER oversampled Improvement model