Automated Identification of Network Anomalies and Their Causes with Interpretable Machine Learning: the CIAN Methodology and TTrees Implementation[1]

Mohamed Moulay[a], Rafael Garcia Leiva[b], Pablo J. Rojo Maroni[c], Fernando Diez[d], Vincenzo Mancuso[e], Antonio Fernández Anta[e]

[a] *University Carlos III of Madrid, Spain*
[b] *Vodafone, Madrid, Spain*
[c] *Nokia Cloud and Networks Services, Madrid*
[d] *Universidad Politécnica de Madrid, Spain*
[e] *IMDEA Networks Institute, Madrid, Spain*

Abstract

Leveraging machine learning (ML) for the detection of network problems dates back to handling call-dropping issues in telephony. However, troubleshooting cellular networks is still a manual task, assigned to experts who monitor the network around the clock. To help in this task we present CIAN (from Causality Inference of Anomalies in Networks), a practical and interpretable ML methodology, which we implement in the form of a software tool named TTrees (from Troubleshooting Trees). We have designed CIAN to automate the identification of the causes of performance anomalies in cellular networks. Our methodology is unsupervised and combines multiple ML algorithms (e.g., decision trees and clustering) and Kolmogorov complexity-inspired data analysis tools that we have developed for this work. CIAN can be used with small volumes of data and is quick at training.

Our experiments use diverse data sets obtained from measurements in operational commercial mobile networks. They show that the TTrees implementation of CIAN can automatically identify and accurately classify network anomalies—e.g., cases for which a network low performance is not apparently justified by operational conditions—training with just a few hundreds of data samples. The resulting information hence enables precise troubleshooting actions. In particular, we showcase how TTrees can be flexibly used to monitor the performance of TCP and QUIC protocols when they are adopted to serve mobile users.

Keywords: Troubleshooting, Anomaly detection, Feature selection, Interpretable machine learning

1. Introduction

Cellular networks have been through an exceptional evolution in recent years. With 4G, and even more with 5G, network services have gained a significant degree of intelligence, involving intensive access to both data communication and computing resources. With the evolution of cellular networks, there has also come an increase in structural complexity and heterogeneity of services, requiring constant monitoring of the communication system. This landscape will only become more complex with the evolution of 5G and the emergence of 6G [2]. Most importantly, new systems will require intelligent and automatic troubleshooting tools, which is the focus of our work.

A line of work on monitoring and troubleshooting of cellular networks is the development of self-healing networks. Self-healing networks are responsible for detecting, identifying, and making decisions on recovery actions [3]. Multiple proposals exist for making fault detection and self-healing systems practical in mobile networks [4]. However, current self-healing troubleshooting proposals lack flexibility and do not scale well. As an alternative, there are newly-defined approaches based on ML, which use deep learning and neural networks as black boxes [5]. Unfortunately, these approaches lack interpretability, so it is not possible to understand the cause of a detected network

[1] A preliminary version of this work appeared in the proceedings of IEEE WoWMoM 2021 [1]

problem, and manual intervention is required for classification after the detection. It is then possible to resort to Explainable AI, or XAI, which deals with the problem of how human users could understand AI's cognition, and decide if an AI based model can be trusted or not [6]. However, XAI does not help interpret AI decisions and conclusions *per se*, although its transparency of operation and interpretability of decisions allows humans to identify which features of the observed system led to specific algorithm's outputs on a case-by-case basis [7]. Multiple methods have been proposed to address the complex issue of ML interpretability, from determining which features contribute the most to a neural network's output, to the development of targeted models that explain individual predictions [8]. However, other studies point out that there is a potential need for applying an interdisciplinary joint efforts in order to correctly apply and leverage explainable/interpretable ML algorithms [9].

In summary, problems like troubleshooting remain a substantially manual procedure, in which highly skilled experts analyze alarms and statistics of performance indicators regularly. These experts need to interpret the outcome of statistical models and ML routines in order to detect and diagnose the cause of problems and anomalies in the network. In contrast, unskilled engineers can not even detect problems effectively [10].

In this paper we focus on detecting and determining the causes of network anomalies automatically. We do not focus on identifying problems for which behavioral models exists or can be built. Instead, we only focus on anomalies that models cannot explain. Specifically, we define an anomaly as a situation that should not happen, given the capacity of the network and the current network traffic. Anomalies are not the same as problems. For instance, while an overloaded base station located in a football stadium full of people yields a network problem, that is not an anomaly, because there is a clear reason for the base station to provide low per-user throughput. On the contrary, an idle base station when the stadium is full is not a network problem (no network alarm is raised), but it is an anomaly, since the base station should be busy. This situation should raise an alarm and call for examination in order to seek for the causes of this anomaly, which might go beyond the network. We are interested in the identification of anomalies because they complement our tools for the identification of problems across the components used to establish a communication between users and servers in a network. Moreover, anomalies are, hopefully, easier to solve than network problems, since they deal with an abnormal operation of the network, not with its limitations.

We go beyond mere anomaly detection and provide an interpretable methodology that can identify the causes of anomalies within the observable features of the system. Our proposed anomaly detection process is applied to data collected in real data from drive test of networks. Specifically, to benchmark QoS in mobile networks, continuous drive tests with end-to-end test scenarios are performed every day internally by the network operators, and externally by third-parties or government regulators. Each of these test campaigns can have up to tens of thousands of individual test cases, from which specific KPIs are calculated. Drive tests are normally executed with off-the-shelf testing equipment (NEMO, TEMS, Swissqual, etc.), capable of running predefined sequences of tests and collecting relevant low level radio and traffic information, as well as application performance statistics. Those test harvest many operational parameters (features) and key performance indicators (KPIs) that allow to monitor network health, so as to evaluate the Quality of Service (QoS) that the network is providing to customers. Features and KPIs identify the operational context of the network and measure the performance of radio, TCP, routing, etc. Hence, we use features and KPIs to detect and classify anomalies, and to trigger troubleshooting actions.

To perform anomaly detection and classification of their causes, we resort to simple and unsupervised ML tools to process the data collected from the network. Using these ML tools, we build an interpretable and robust methodology, named CIAN, which we have implemented as a software named TTrees and tested with real data. In addition to interpretability, a key advantage of CIAN and TTrees is that it does not require access to large volumes of training data.

1.1. Original Contribution of the Work

We propose an unsupervised methodology for the detection and classification of network performance anomalies, which we call Causality Inference of Anomalies in Networks (CIAN). In addition, we provide a software implementation of the methodology named Troubleshooting Trees (TTrees). Hence, with this work, we join the ML research stream in networking, while focusing on the troubleshooting of possible network issues even with small volumes of measurement data. However, differently from existing ML proposals, we detect anomalies by simply identifying the scenarios that ML algorithms *cannot* learn/explain. For instance, a network that shows low throughput is not anomalous if its low performance can be explained, e.g., by detecting the presence of a bad radio link. For this reason, a novel aspect of our work is the use of simple interpretable ML algorithms, moving away from the current trend

of high-accuracy ML algorithms (e.g., deep learning) that do not allow interpretation (and hence understanding) of their outcome. Specifically, we use decision trees in our methodology since they are transparent to inspection, hence interpretable [11]. In fact, having low accuracy ML models in not a problem to our system, since we consider as interesting (anomalous) the scenarios that are misclassified by the ML algorithms, and we do not want to miss them by overfitting. Then, the anomalous scenarios are classified by the potential causes of their behavior. Understanding and classifying these anomalous scenarios allow alerting the appropriate department to take corrective actions.

In order to evaluate the methodology, we use real operational network data. Two of the datasets used here were collected for cellular service auditing purposes by Nokia in various European countries. They include thousands of features (i.e., operational parameter reports and statistics) and a few KPIs used to mark the quality of the network in each test (i.e., throughput, connection establishment time, download time, etc.). However, the available datasets are also quite limited in terms of number of data samples, e.g., they only report hundreds or a few thousands of experiments each. Thus, these datasets are not suitable for commonly adopted deep neural networks and ML methods requiring complex training. The other datasets have been obtained in measurement campaigns conducted by us with MONROE, an open-access platform for multi-homed experimentation with commercial mobile broadband networks across Europe [12]. With MONROE, we have access to larger datasets, although with a reduced number of features with respect to Nokia's datasets. Moreover, while Nokia's datasets report mostly TCP traffic, with MONROE we were able to experiment with TCP and QUIC. As a result, in this article we show how CIAN can be flexibly used to study heterogeneous and diverse datasets to identify the root causes of anomalies without human intervention.

The experts in detection and classification of network issues from the research team have been instrumental in the validation of the methodology and the TTrees system. They have manually inspected the outcome of the unsupervised process, verifying that the scenarios that were classified as anomalous did in fact show strange combinations of observed features and KPIs, and that their unsupervised classification into *network aspects* (i.e., combination of observed operational features) that identify the causes of the anomalies was consistent. Additionally, the experts guided the development of a supervised implementation of the methodology, STrees, in which features were manually aggregated into a few relevant network aspects based on the experts' knowledge and experience, like it happens in currently implemented troubleshooting protocols. The comparison of the outcomes of unsupervised and supervised methods with the same data has also been used to validate the methodology.

As mentioned, we provide a Python open-source implementation of our methodology based on the Scikit-learn library [13] and a new library that we have implemented.[2]

In this extended version of the paper, we carried out an in-depth evaluation of the methodology and its implementation through the execution of experiments in cellular networks. More especifically, we expanded the related work by introducing additional references on causality and explainable AI with the current state of the art, introduced and analyzed two datasets with imbalanced anomalies and tested an alternative implementation of CIAN with a new clustering algorithm. We used this alternative implementation to draw performance comparisons that assess the shortcommings of TTrees and propose possible solutions to the latter.

1.2. Organization of the Paper

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 overviews CIAN, the unsupervised methodology proposed. Section 4 explains in detail how this methodology has been impemented as the TTrees system. Section 5 presents and analyzes the results TTrees achieves when it is applied to real datasets of various kind. Finally, we summarize and conclude the paper in Section 6.

2. Related Work

2.1. Causality

Causality refers to the relationship existing between two types of events, cause and effect, where the occurrence of the latter is a consequence of the occurrence of the first. Causality must not be confused with another common term belonging to the statistics lingo, which is correlation. While the latter refers to the relationship between two variables

[2] `https://github.com/Mohmoulay/WoWMoM2021`

based on their covariance (if a variable changes, the other does so too), causality establishes a clear dependency and a mapping in the way these changes take place. Causality always implies correlation but not the other way around [14].

Traditional methods for the discovery of causal relationships include the conduction of randomized trials with the objective of removing confounders, that is, variables that may deceive into drawing non-existent associations between different events. Due to the high cost of implementing these experiments, many researches have switched to the discovery of causal relationships through the exploratory analysis of observational data. This is known as causal inference [15]. As a consequence, and due to the exponential increase in the amount of data that can be collected and analyzed by systems nowadays, machine learning has been put at the forefront as a promising pathway to the automatization of causal inference and its execution in a computationally feasible amount of time. However, the introduction of machine learning models can lead to the so-called "black box effect," as the convoluted nature of these algorithms can obscure the understanding of the reasons behind the cause-effect connections and predictions discovered [16].

Advances have been done theoretically to favour the explanation of these causal relationships in the field of machine learning. Judea Pearl proposed a three-level hierarchy to classify causal information according to the type of queries each family can give answer to [17]. Firstly, the Association level, at the bottom of the causal hierarchy and syntactically expressed through conditional probability sentences, deals with information that allows to infer data associations extracted from the pure observation of data and the application of standard probabilities. Secondly, the Intervention level, regards questions that not only imply the observation of data but also the alteration of the perceived reality. The probabilistic expressions at this level can be estimated both through experiments and analysis using causal Bayesian networks. Finally, at the top of the hierarchy we have the Counterfactual level, which deals with retrospective reasoning and the hypothesis of scenarios provided environmental conditions had been different or altered prior to observation.

Pearl's three-tier hierarchy sheds light on why most machine learning systems, which are only capable of extracting data associations and thus fit in the bottom of the causal hierarchy, are unable to reason about newly unobserved data and provide causal explanations. Thus, this raises the question whether artificial intelligence can be enriched to provide new layers of causal inference. The main pathway towards a second generation of machine learning models capable of achieving this level of cognition involves the development of Explainable AI (XAI).

2.2. Explainable AI

Using Machine learning to detect anomalies has been around for a while. Currently, the most powerful algorithms are based on Neural Networks, which show high accuracy but have little interpretability. In reality, interpretability mainly refers to the intuition behind the outputs of a model; the more interpretable a machine learning system is, the easier it is to identify causality within the system's inputs and outputs. Thus, XAI has gained much momentum in the past few years to help explain any black-box Model. The most complete techniques to achieve that are the local interpretable model-agnostic explanations (LIME) and Shapley Additive explanations (SHAP), based on the current state of the art.

The LIME [18] method is one of the most popular interpretability methods for black-box models. Following a simple yet powerful approach, LIME can generate interpretations for single prediction scores produced by any classifier. For any given instance and its corresponding prediction, simulated randomly-sampled data around the neighborhood of the input instance for which the prediction was produced are generated. Subsequently, new predictions are made for the generated instances and weighted by their proximity to the input instance while using the model in question. Lastly, a simple, interpretable model, such as a decision tree, is trained on this newly-created dataset of perturbed instances. By interpreting this local model, the initial black-box model is consequently interpreted. Although LIME is powerful and straightforward, it has its drawbacks. In 2020, the first theoretical analysis of LIME [18] was published, validating the significance and meaningfulness of LIME and proving that poor parameters choices could lead LIME to miss out on essential features.

SHAP [19] is a game-theory-inspired method that attempts to enhance interpretability by computing the essential values for each feature for individual predictions. Firstly, the authors define the class of additive feature attribution methods, which unifies six current methods, including LIME [18], DeepLIFT [20], and Layer-Wise Relevance Propagation [21], which all use the same explanation model. Subsequently, they propose SHAP values as a suitable feature importance measure that maintains three desirable properties: local accuracy, missingness, and consistency. Finally,

they present several different methods for SHAP value estimation and provide experiments demonstrating the superiority of these values in terms of differentiating among the different output classes and better aligning with human intuition than many other existing methods.

These techniques are generic and help interpret models deployed in multiple disciplines, such as healthcare, medicine, retail, and banking. A key factor concerning these techniques is that they have to be used on pre-trained models. We differ because we are trying to deploy techniques that are easy to interpret throughout the entire process, from data processing to problem classification.

2.3. Anomaly detection

Early works on fault detection suggested the use of time series *regression* methods and *Bayesian networks*. For instance, Barco *et al.* [22] produced an automated tool for troubleshooting mobile communication networks back in the days of 2G, relying on Bayesian networks to detect call drops. Khanafer *et al.* [23] followed up on proposing a method based on Bayesian networks to detect faults in UMTS systems, in which they apply different algorithms to discrete KPIs. Other works, such as [24], rely on a scoring-based system, in which the authors build the fault detection subsystem around labeled fault cases. These cases were previously identified by *experts*, using a scoring system to determine how well a specific case matches each diagnosis target. The work presented in [25] is based on a supervised genetic fuzzy algorithm that learns a fuzzy rule base and, as such, relies on the existence of labeled training sets. Indeed, most of the techniques proposed in the literature focus on using supervised machine learning algorithms [23, 24, 25]. In this paper we show that it is convenient to use unsupervised techniques to unveil hidden information in the input data, without restricting a priori the possible outcomes.

Other works make use of advanced mathematical and statistical tools. For instance, Ciocarlie *et al.* [26] address the problem of checking the effect of network changes via monitoring the state of the network, and determining if the changes resulted in degradation. Their fault detection mechanism uses *Markov logic networks* to create probabilistic rules that distinguish between different causes of problems. A framework for network monitoring and fault detection is introduced in [27], using *principal component analysis* (PCA) for dimension reduction, and kernel-based semi-supervised fuzzy clustering with an adaptive kernel parameter. To evaluate the algorithms, they use data generated by means of an LTE system-level simulator. The authors claim that this framework proactively detects network anomalies associated with various fault classes. These methods lack the flexibility of ML-based ones and, differently from our proposal, cannot be fully automated for a generic network context.

Recently, researchers are exploring the potentials of AI/ML for predicting and timely obviating network performance issues. For instance, Terra *et al.* [28] analyze how to apply existing XAI methods to identity the root causes of service level agreement violations in a sliced network, in a 5G context. Tang *et al.* [29] use instead ML to identify the root causes of exceptions in packet-over-optical networks, thus achieving what they define customized performance monitoring and troubleshooting. Furthermore, Wei *et al.* [30] point out that the class of AI that goes under the label of intent-based networking techniques can be used for troubleshooting of network services. For instance, available products like Cisco ACI and Spruce Network DeepFlow already offer network operators the capability to monitor and rise alarms and trigger troubleshooting actions in 5G and beyond 5G networks. However, they do not offer explainable nor interpretable methods and basically help in the self-optimization of a system using intent-based networking, which is limited to the scope of SDN/NFV network functionalities, while we will see in the reminder of this article that contextual information is key to identify anomalies rather than generic performance issues, and their causes. Moreover, existing AI/ML based approaches to troubleshooting are highly customized (e.g., for 5G functionalities or for optical networks) and require the collection of specific network features. Differently from our approach, these limitations make existing approaches unsuitable to be flexibly applied to new protocols and services.

Finally, it is worth mentioning that most of the existing proposals have been evaluated only through simulators, and require large datasets. They help to detect network issues, but do not contribute to interpreting the network behavior. By comparison, in our work, we use not-necessarily-abundant data collected in real operational networks and propose a fully automated ML-based methodology that leads to a straightforward interpretation of network behaviors. This includes identifying not only the occurrence of problems but also their root causes.

3. Overview of CIAN

In this section, we briefly describe the framework, and the steps into which the automated unsupervised methodological process we propose is divided, without entering in the details of design and implementation, which will be instead the object of Section 4. We begin by defining some basic notations and concepts that will be used to describe the methodology. The basic notation used is summarized in Table 1.

Table 1: Basic notation.

Symbol	Description
M	Model
D	Dataset
$L(M)$	Model length
$L(D\|M)$	Length of the dataset D given the model M
$\mathbf{x}$	Collection of data samples
$\mathbf{x}_i$	A specific data sample
x_{ik}	A specific feature of a data sample
$\mathbf{y}$	Set of target KPIs
m	Number of classes resulting from KPI discretization
C_1	Initial classifier (Knowledge Tree)
A	Set of anomalous scenarios
$\mathbf{x}^A$	Set of features in anomalous scenarios
$\mathbf{y}^A$	Set of target KPIs in anomalous scenarios
R	Set of features that are highly informative about the target KPI
r_a	Subset of relevant features for problem classification
α	Number of subsets selected for problem classification
C_2	Second classifier (Aspect Classification Tree)

3.1. The Core Idea for Detecting Anomalies

Our methodology has been developed to find and interpret anomalies in network communications, in an automatizable way. The core idea behind our approach has been inspired by the concepts of *Kolmogorov complexity*, and in particular, by the *minimum description length* (MDL) and *minimum message length* principles [31, 32]. According to these concepts, learning from data is equivalent to find regularities, or equivalently, compress data, i.e., describe data in the shortest possible way without loosing information. In particular, the MDL principle claims that the best model M for a dataset D is the one that minimizes the following expression.

$$L(M) + L(D|M), \tag{1}$$

where $L(M)$ is the length of a model M encoded as a string of symbols, and $L(D|M)$ is the length of the dataset given the model. Intuitively, this latter term can be seen as the length of the list of errors made by the model M when predicting KPIs from features with a specific dataset D, encoded using an optimal length code. In this sense, we divide our dataset into two disjoint groups: the compressible part of D that can be described through M, and the incompressible part that includes all the points in D that are wrongly predicted (e.g., misclassified) by M. In a network data context, the compressible part of D is therefore what the model M identifies as regularities in the network described with a given set of features and KPIs, while the incompressible part corresponds to unexpected behaviors for which the KPIs cannot be predicted based on the observed features. In general, the more complex we allow the model to be, the more regularities we will be able to find, but also the longer its representation will be; thus, the appropriate balance between model complexity and model accuracy has to be found, so as to minimize expression (1). In this paper we address this trade-off between model complexity and model accuracy by means of applying an incremental procedure over model hyper-parameters to find the most complex possible model that does not overfit the dataset. The particular families of models used are selected for their accuracy and interpretability, i.e., they produce models that identify the compressible part of D based on interpretable rules. From here it follows the claim that the incompressible part of D cannot be interpreted, and hence it contains *anomalies*.

3.2. Input

The input to the process is a collection $\mathbf{x}$ of data samples, obtained from n network operation scenarios. Each data sample $\mathbf{x}_i$ includes k features $(x_{i1}, \ldots, x_{ik})$, which are operational characteristics of the network and the context in which measurements are taken, in scenario $i \in [1, n]$. In addition to these features, there is a distinguished set of ℓ target KPIs $\mathbf{y}$ which drives the search for anomalies. Hence, the dataset is formed by $\mathbf{x}$ and $\mathbf{y}$, and each scenario $i \in [1, n]$ is a row of the dataset containing features $(x_{i1}, \ldots, x_{ik})$ and KPIs $(y_{i1}, \ldots, y_{i\ell})$. Note that realistic datasets can contain incomplete data entries. Therefore, before processing further a dataset, we will possibly filter out malformed or incomplete rows, and all rows that do not contain values for the target KPIs.

3.3. Discretization

The first step of our methodology consists in grouping data samples into classes, based on the value of the target KPIs $\mathbf{y}$. The reason for having this step is because, in general, the target KPI values belong to an infinite domain.[3] Note that our methodology is oblivious to the process that collects data samples, as we simply take a dataset and work with all its valid entries. For discretization, it is desirable that the number of classes m used is manageable but not too small, for expressiveness. However, we do not need vast amounts of data, as our methodology works with as few as tens of samples per discretization class, so as to have a bare minimum level of statistical relevance. Additionally, it is desirable that data samples are reasonably balanced across classes, although this may not be perfectly achievable in all cases. Finally, in order to provide semantics to the discretization, every vector of KPIs $(y_{i1}, \ldots, y_{i\ell})$ is assigned a certain QoS, which can be quantified. Then, the discretization places in the the same class scenarios whose QoS are similar, and the m classes can be ranked by an expert based on their goodness (e.g., as "very bad", "bad", "medium", "good", "very good", etc.).

3.4. Selection of Anomalous Scenarios

Once the data samples have been assigned to the classes based on the value of the target KPIs, we use them to train a classifier C_1. This classifier will use the features $(x_{i1}, \ldots, x_{ik})$ to determine whether, in scenario i, the KPIs $(y_{i1}, \ldots, y_{i\ell})$ seem to belong to class $j \in [1, m]$. The objective is to build a classifier that is able to correctly classify as many data samples as possible without overfitting. It is also desirable that the decisions of the classifier can be interpreted.

Applying this classifier C_1 to all the data samples will hence incorrectly classify a subset of them, out of which we consider the subset $A \subset [1, n]$ for which the classifier predicts classes higher than the observed ones. The scenarios in set A are the ones that we consider interesting, or *anomalous*: scenarios whose target KPIs is overestimated with a properly trained classifier C_1.[4] The interpretability of the classifier helps to determine why the data samples in A are anomalous (with respect to those properly classified). This means identifying which features are relevant and how to differentiate A from the rest.

Observe that the scenarios in A are not necessarily those in which any of the target KPIs show low performance. For instance, if $\mathbf{y}$ consists of a single KPI (e.g., the average throughput observed), a scenario i with low throughput y_i may not be anomalous if this is due to a low radio coverage (i.e., when the user is at the cell edge). This scenario should be correctly classified by an interpretable classifier C_1 as bad, and a visual inspection of the radio features in $\mathbf{x}_i$ should reveal the issue. The anomalous scenarios that are of interest to us are those that cannot be (directly) explained. For instance, a scenario i with low throughput y_i in which the features $\mathbf{x}_i$ are all good.

Hence, from this point on, our target will be to identify automatically which types of anomalies the scenarios in A show. This could later be used to report the issue (complemented with the data samples) to the appropriate department, that may take corrective action.

[3] We explored options without discretization, but they were unsatisfactory, because they led to complex approaches with additional hyperparameters.

[4] In some practical cases, it may be interesting to consider in the set A also the scenarios for which the classification assigns a worse class (in QoS terms) than their actual class. Anomalies that have a positive impact on QoS are not harmful at all for the user and could be treated less urgently than anomalies with negative impact on QoS, so we ignore them in this work. Still, they would deserve further studies because they could reveal the existence, and possibly explain the cause, of new interesting effects to be considered for design and tuning of network systems and services. The methodology described in this article can be straightforwardly extended to analyze also this kind of anomalies.

3.5. Selection of the Most Relevant Features

Let $\mathbf{x}^A$ and $\mathbf{y}^A$ be the set of features and corresponding KPIs in anomalous scenarios, taken as $\mathbf{x}_i$ and $\mathbf{y}_i$, for $i \in A$. These are the scenarios that we want to explore further. A natural approach to attempt for understanding what makes these scenarios anomalous would be using another classifier only for them. While this indeed separates these scenarios into different classes, our experience has shown that the results are however hard to interpret, even by an expert.

For this reason, for the sake of interpretability, in our methodology we proceed by restricting the set of features for further analysis. Hence, in the next step of the methodology, we select a small subset $R \subset [1, k]$ of features. The set R should contain only non-redundant features, and should contain those that are highly informative with respect to the set $\mathbf{y}^A$. The process we apply to select R is as follows. First, rank the features of $\mathbf{x}^A$ by the amount of information they provide about $\mathbf{y}^A$. Then, using this ranking, select the top features in order, removing those that are redundant with respect to the previously selected features. This process stops when a certain number of features have been selected, or when the next feature in the ranking gives little information about $\mathbf{y}^A$.

3.6. Clustering Using the Most Relevant Features

The next step of the methodology is to use the features in R to classify the anomalous scenarios into meaningful types. We have observed empirically that the features in R naturaly belong to different network aspects (e.g., radio features, TCP features, QUIC features, etc.), and that an anomalous scenario may show issues in several of these aspects simultaneously. This leads to a process in which subsets $r \subseteq R$ are selected, and each subset is used to classify the anomalous scenarios using binary clustering indicating whether the selected subset of features seems to be (at least partially) responsible for the anomaly. The conjecture is that this (unsupervised) process will select subsets that correspond to different network aspects, and the clustering will separate "good" scenarios from those with issues in that aspect. While this process is unsupervised, our experience is that it leads to subsets of R that an expert can easily match with particular aspects of the network.

Hence, in this step we select α subsets $r_a \subseteq R$, $a \in [1, \alpha]$, and apply each of them to divide the set A of anomalous scenarios into two clusters, a *low-performing* cluster (or a cluster with issues with respect to that network aspect, indicated with a bit 0) and a *high-performing* cluster (bit 1). The bit labels 0 and 1 are assigned automatically to the clusters, depending on which of them has better performance (in terms of QoS). Moreover, it is desirable to have balanced, compact and non-redundant clusters. Each anomalous scenario will be assigned to one of the two clusters generated with each subset r_a. This can be represented as a binary string of length α, with a value 0 or 1 in each position, which classifies all anomalous scenarios A into 2^α types. Each of the bits in the vector associated to a scenario conveys whether that scenario shows issues in the particular network aspect.

3.7. Aspect Classification of Scenarios

Finally, a second classifier C_2 is built using $\mathbf{x}^A$ and $\mathbf{y}^A$ as training data, and the 2^α types as the classes to which these scenarios are assigned. Observe that we do not restrict the classifier C_2 to use only the features from R (although it is very likely that they will be used). Thus, C_2 uses for training $\mathbf{x}^A$, $\mathbf{y}^A$, and the α-bit string of every scenario in A (identifying the presence of anomalies in the α selected network aspects, as described in Section 3.6). The output of C_2 will group anomalies into 2^α classes, each with a specific anomaly pattern.

It is important that this classifier is interpretable, because this makes experts able to understand why an anomalous scenario has to be considered to have issues or not in each of the automatically identified network aspects. Note that if an expert had been involved in the most relevant feature selection process, she should have assigned a specific network aspect to each subset $r_a, a \in [1, \alpha]$, and possibly a meaning to every binary value of every element of the network aspect anomaly string. Be it the output of manual inspection or automatic data analysis, this information can be used to determine the issues each particular anomalous scenario has, and can be sent to the appropriate department to promptly intervene.

3.8. Using the Classifiers to Detect and Identify Anomalies

After the previously described steps have been completed, we found ourselves with two classifiers C_1 and C_2, that can be used to detect anomalous scenarios, determine network aspects that make this scenario anomalous, and hence notify about this to the appropriate department for troubleshooting. Now we can use these trained classifiers to analyze fresh data inputs and detect anomalies. In particular, consider a new scenario $(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$. Using

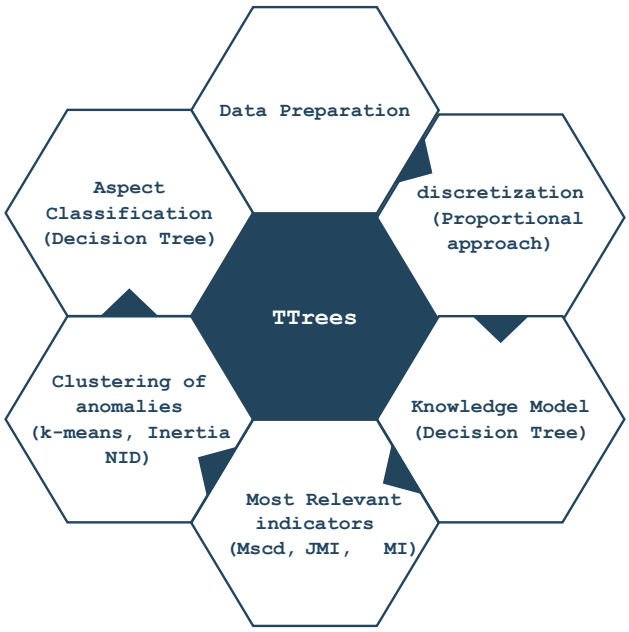


Figure 1: Steps of TTrees (using multiple ML techniques): beginning with data preparation, proportional discretization, training of knowledge tree, selection of the most relevant features, identification of anomaly clusters, and training of a network aspect anomaly classifier.

classifier C_1 we determine that this is an anomalous scenario if the KPI class C_1 assigned to $(x_1, \ldots, x_k)$ is not the one corresponding to the discretization class of $(y_1, \ldots, y_\ell)$ but it is actually better. If that is the case, C_2 will be used, so that the class assigned to $(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$ is an α-bit string expressing whether this scenario has issues in each of the α different network aspects or not, which allows for alerting the appropriate department(s).

4. Implementation of the CIAN methodology

Here we present the implementation details of the tool TTrees that automatizes CIAN, the methodology presented in the previous section. Fig. 1 depicts the different phases of the proposed methodology according to their implementation in TTrees. As reported in the figure, our implementation leverages a number of standard tools for data analysis and ML, from discretization algorithms to clustering and classification.

4.1. Data Preparation in TTrees

The first phase consists in preparing the available data. This is a necessarily preliminary step that includes data filtering, cleaning of the dataset from the presence of incomplete entries, and extracting the k features in $\mathbf{x}$ and the target KPIs in $\mathbf{y}$ that need to be observed (i.e., note that the given dataset might include more features and KPIs than what we are interested in). Thus, we use a Python script to extract $\mathbf{x}$ and $\mathbf{y}$ from the dataset. The script automatically discards entries in which one or more KPIs or features are not reported. It also discards features that are constant through the dataset, which would not be informative. Features can be either numeric values or categorical entries (e.g., labels used to indicate the operational conditions of the network, like the name of the radio technology used, the name of the protocol adopted for transmission, etc.).

4.2. Discretization in TTrees

The selected target $\mathbf{y}$ is usually a set of one or more numerical KPIs, whose values may lay on a continuous real interval. Thus, a first problem to address in the training of TTrees consists in discretizing the continuous target into a finite number of classes (i.e., categories). Using a simple quantization of the values in $\mathbf{y}$ would introduce an error in the subsequent data analysis. Hence, instead we keep the target values as they are in the dataset, and assign a label

to each point, which identifies their class. Since TTrees is unsupervised, discretization uses progressive numbers as labels (e.g., "0" to "5").

In our methodology, being based on the study of the compressibility of the training dataset using optimal lengths codes, it is of upmost importance that the discretization algorithm applied does not alter the distribution of the training samples, that is, both the continuos KPI and the discretized version should follow the same distribution. A uniform discretization is an easy to implement approach that satisfies this requirement. Therefore, we adopt a uniform discretization of target KPIs.

The number of classes (labels) used is not pre-defined. Instead, it is automatically derived by TTrees from the available number of points in the dataset. There exist a collection of candidate discretization algorithms that are not biased and have low variance (e.g., equal width, equal frequency, or fixed-frequency). Unfortunately, they require optimizing hyperparameters [33], which makes them inadequate for an unsupervised automated software tool. Instead, we use a discretization approach that does not require any hyperparameter tuning: *proportional discretization* [34]. This method uses a number of categories proportional to the size of the dataset. The number of categories used, denoted by m, is computed as $m = (\log_2 n)/2$, where n is the number of samples of KPIs $\mathbf{y}$. We use the proportional discretization approach to identify the centroids of the m intervals of KPI values (associated to the m categories). After identifying the centroids, we apply k-means clustering to the continuous target variable $\mathbf{y}$, which assigns labels to the data points and returns the boundaries between categories intervals.

4.3. Knowledge Model for Identifying Anomalies

Once the data points $\mathbf{y}$ are assigned to a finite number of categories, TTrees continues by building a knowledge model, meant to identify anomalies. For this step we use an ML classifier, which we want to be interpretable, so as to allow the user to understand what kind of anomaly is detected. This is achieved by training a decision tree [35] with the data set $\mathbf{x}$, and using as classes the categories defined in the discretization phase. The decision tree infers the class for a sample $\mathbf{y}_i$ of $\mathbf{y}$ from the values of the associated features $\mathbf{x}_i$. This justifies the use of terms "knowledge model" and "knowledge tree" for the role of this classifier. However, some samples $\mathbf{y}_i$ can be misclassified, representing anomalous behavioral trends that cannon be learned. Note that, with more advanced models (neural networks, random forests, etc.) instead of a decission tree, we could gain accuracy, but we would lose interpretability and the model may discard interesting samples.

The knowledge tree is build by applying the widely adopted Classification And Regression Tree (CART) algorithm [36], with the Gini impurity metric [37] computed by comparing the output of the classification with respect to the discretization categories. We refer the resulting decision tree as C_1. The depth of C_1 is limited to a maximum of $\lfloor(\log_2 n)/2\rfloor$, so that, on average, the leaves of the tree hold a sufficiently large number of samples to properly classify the KPIs. A deeper tree would suffer a high risk of overfitting [38]. Moreover, we also limit the number of samples at the tree leaves to a minimum of five, to further reduce the risk of overfitting [37].

After the full tree has been derived to the maximum depth allowed, we perform a cost-complexity pruning, which is the best known approach to avoid overfitting in potentially large trees [37], combined with cross-validation of the candidate pruning points. The leaves of the tree that have the highest Gini impurity measure are removed first. Pruning minimizes a cost-complexity metric that linearly combines the classification error (cost) made by the tree and the size of the tree (complexity). However, the pruning metric has a hyperparameter which represents the relative importance of cost with respect to complexity. Cross-validation is used to tune that hyperparameter automatically. In practice, for each candidate value of the hyperparameter, we identify the tree with the minimal cost-complexity, and cross-validate the performance of that tree by splitting the dataset into smaller pieces of at least 30 samples each: we use all but one of the pieces to train the tree and the remaining one for evaluation. This process is repeated as many times as the number of pieces in which we break the original dataset (so as to use every piece for validation exactly once).

Once C_1 is created, for each sample of $\mathbf{y}$ we have two possibilities: the sample is classified or not under the same class in the discretization phase and by the knowledge tree. If the knowledge tree returns a class higher than the one of the discretization phase, an anomaly of interest is identified and included in set A.

4.4. Most Relevant TTrees Features

The next step consists in identifying a subset of features that are relevant for the anomalous scenarios in A, so as to identify which network aspects are relevant to explain the anomaly. TTrees then evaluates each of the available

features by computing how much information on the target $\mathbf{y}$ is contained in the feature. For this task we use the well known metrics *mutual information* (MI) [39] and *joint mutual information* (JMI) [40], from classic information theory.

We have also tested a novel metric that we have built on top of the concepts of Kolmogorov complexity and normalized compression distance (NCD) [41], which we call *miscoding* (Mscd). The miscoding of a sequence of p different features $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_p\}$ with respect to the target $\mathbf{y}$ is defined as

$$\mathrm{Mscd}(\mathbf{x}_i, \mathbf{y}) = \frac{1 - \mathrm{NCD}(\mathbf{x}_i, \mathbf{y})}{\sum_{j=1}^{p} \Big(1 - \mathrm{NCD}(\mathbf{x}_j, \mathbf{y})\Big)}. \tag{2}$$

We propose Mscd because it measures the difficulty of reconstructing the target $\mathbf{y}$ from the features of $\mathbf{x}$ and vice versa, which accounts for the redundancy in $\mathbf{x}$. In TTrees we start by computing a conditional redundancy matrix with the normalized compression distance of all possible pairs of features with respect to the target variable $\mathrm{NCD}(\langle \mathbf{x}_i, \mathbf{x}_j \rangle, \mathbf{y})$, based on a joint discretization of the attributes, and the computation of optimal length codes, given the relative frequencies of the discretized vectors. Then, we select the attribute with the highest NCD, and recompute the values of the redundancy matrix assuming that this value is selected. We repeat this process of selection of the best feature and recalculation of the redundancy matrix until all the features have been selected. The final miscoding is given by normalizing over one minus the NCDs computed for the different attributes.

MI is applied on a per-feature basis and is able to quantify the relevance of a feature, but it fails to identify redundancy. JMI is instead used on feature pairs, so that it allows to evaluate not only the relevance of an feature but also the redundancy with another feature. However, JMI is prone to errors in case outliers are present. Mscd offers the possibility of evaluating relevance and redundancy, plus the quantity of irrelevant information contained in the feature.

We will compare the performance obtained by applying MI, JMI and Mscd in the numerical evaluation section. Here it is enough to mention that in all of the three cases, we sort the features in decreasing order according to the selected metric, and pass the top of the list to the next TTrees phase. In particular, since we will need to cluster the anomalies based on selected network aspects, we pass to the clustering phase a number of features much larger than the number of aspects to study. For this paper we have used m^2 features, where m is the number of bins in which the target KPI has been discretized.

4.5. Clustering of Anomalies in TTrees

Each *pair* of relevant features is regarded as a potential *network aspect*. For each aspect, TTrees applies a clustering algorithm on all anomalous samples. This bidimensional clustering is done with the k-means algorithm [37], using the normalized values of the features obtained via RobustScaler [42]. In addition, and for visualization purposes only, when we have just one target KPI, we use a simple regression with respect to $\mathbf{y}$ on each of the features, and we sort samples according to increasing regression values. This allows to plot anomalous samples consistently so that higher $\mathbf{y}$ values are at the top-right corner of the plots. As a consequence, the cluster of samples at the top-right corner contains the samples with highest $\mathbf{y}$ values and the cluster at the bottom-left corner the samples with lowest $\mathbf{y}$ values. If the target KPI $\mathbf{y}$ is better when larger (resp., smaller), then the bottom-left (resp., top-right) cluster is the one with issues in the network aspect defined by the two features. In general, the cluster with issues is assigned a label 0 in this aspect, and the other cluster is assigned 1.

The problem is that we have potentially a large number of pairs of features (and hence network aspects), and some of them might bring redundant information. Therefore, TTrees selects only a few pairs of features as network aspects, so as to easily and interpretably identify non-redundant descriptions of anomalies. In our current implementation we select $\alpha = 4$ feature pairs, i.e., the 4 most relevant aspects to evaluate to understand the anomaly. Thus, the number of features passed from the previous step is $p = \alpha^2 = 16$. We do not use all possible features for a matter of practicality. For example, if we had $p = 120$ features, which is a reasonable value for cellular data traffic traces, we would have to evaluate $p(p-1) = 14280$ pairs, and make hundreds of millions of comparisons to test the redundancy among pairs.

The most relevant and descriptive feature pairs are selected based on multiple metrics. First of all, TTrees computes the inertia score [43] of the clusters resulting from each feature pair. The inertia of a cluster quantifies the tightness of the clusters (the lower, the better). The resulting ranked list is filtered by using two criteria: TTrees only

keeps clusters with low redundancy and which are balanced in the number of elements in the two clusters. Redundancy of clusters is computed using the normalized information distance metric (NID) [44]. Here the order matters, since if two feature pairs yield redundant clusters, the feature pair with lower inertia is retained, while the other is discarded. In order to automatically tune and apply both filters, we select filtering thresholds with the help of the histograms obtained for the distributions of redundancy scores and balance metric. We have observed that the histogram of redundancy shows a bimodal distribution, therefore we select as threshold to accept/reject a feature pair the point with the lowest value in-between the two peaks of the bimodal distribution. For balancedness, the distribution histogram is multimodal, with an automatic threshold selection where the minimum point between first and second peaks (minimum accepted value of balancedness), and the minimum point between the last two peaks (maximum accepted value for balancedness).

4.6. Aspect Classification in TTrees

The α top feature pairs, obtained in the previous phase of TTrees, identify network aspects in which the anomaly could be rooted. In this phase they are used to train an interpretable classifier (again a decision tree) to map the anomalies A onto the 2^α classes corresponding to the combinations of the α network aspects relevant for the anomalies. This *aspect classifier*, denoted as C_2, uses the the binary values of the aspects assigned in the previous phase, and builds new categorical target variables consisting of binary strings that combine these binary aspect values. The elements with value 1 in the binary string identify aspects in which the scenarios of that category may have issues, and for which a network specialist should be called. Note that this classifier makes decisions based on the full list of features, and not only based on the most relevant features identified by TTrees. In the training of C_2, like for C_1, we avoid overfitting by limiting the depth of the tree to $\lfloor (\log_2 |A|)/2 \rfloor$, and apply cost/complexity pruning with cross validation to increase the generalization level of the tree.

4.7. Software Implementation

TTrees is implemented using Python. We use two libraries: (*i*) the widely adopted scikit-learn library, which provides us with ML algorithms, such as decision trees and k-means, and (*ii*) a library developed by us for automatic ML tools, which includes an Mscd calculator.

5. Empirical Evaluation

Existing troubleshooting tools do not generalize very well and need the assistance of an expert. For this reason, here we evaluate TTrees along with a supervised version of our tool, which provides a validation baseline. For simplicity, here we use only one target KPI (e.g., the throughput) and restrict the analysis of anomalies to the case in which the knowledge model predicts better throughput than what observed. We also present a modified version of TTrees, in which we modify the clustering algorithm in order to effectively analyze imbalanced datasets. The first datasets used in this section were extracted from experiments executed in controlled environments. These are complemented with Google and Facebook datasets, which are the result of homogeneous cellular network conditions and include nodes deployed in both static and mobile stations. The latter can suffer from network outages and undergo unexpected variances in the quality of the signal, which enriches the possible scenarios captured in the dataset and can be a clear source of anomalies. As it will be later developed, these uncertain conditions yield aspect classification clusters where the number of anomalies is imbalanced, that is, the ratio is not equal between clusters and one class accumulates most of the samples.

The datasets collected for this paper and the analysis of all datasets presented here, for all services and operators discussed, are available in GitHub.[5]

5.1. Supervised ML-based Troubleshooting

The supervised version of TTrees, denoted as STrees, uses a supervised selection of network aspects that are most relevant to identify and cause anomalies in the problem classification phase of the methodology. In particular, with the supervision of a network expert, STrees selects as network aspects those combinations of features that are most relevant for the selected target KPIs, as identified by the expert. Afterward, in STrees, we take the tagged classes and use them to train C_2, i.e., the final aspect classification tree, like is done in TTrees.

[5] https://github.com/Mohmoulay/WoWMoM2021

Table 2: Brief description of the experimental datasets used in this work

ID	Dataset	# Samples	# features	Families of features
#1	**HTTP FILE DL**	6,690	228	Radio, RTT, Duration, TCP Volume, TCP Flags, TCP Window, Packet Anomaly
#2	**HTTP LIVEPAGE DL**	10,500	126	Radio, RTT, Duration, TCP Volume, TCP Flags, TCP Window, Packet Anomaly
#3	**HTTP FILE DL MONROE**	120,000	106	Radio, RTT, TCP Window, Packet Anomaly
#4	**QUIC FILE DL MONROE**	3,951	91	RTT, Duration, TCP Volume, TCP Flags, TCP Window, Packet Anomaly
#5	**Facebook Page Download**	1447,839	109	Radio, RTT, TCP Window, TCP, Packet Anomaly
#6	**Google Page Download**	1261,119	109	Radio, RTT, TCP Window, TCP, Packet Anomaly

5.2. Datasets

Table 2 summarizes the characteristics of the datasets used in this work. We first analyze two data collections of cellular network experiments used by Nokia in 2019 for network performance assessment of various European 4G networks (Datasets #1 and #2). These two datasets contain drive-test information on TCP traffic experiments in which constant-size (3 MB) files are downloaded, and live web-pages are fetched with a mobile user device. Then, we further evaluate TTrees using larger mobile broadband connectivity datasets collected with MONROE [45]. This includes generating a dataset of QUIC *qlogs* by running experiments from MONROE nodes.

More in detail, the Nokia datasets used for experimental validation are real test records generated by processing all the information provided by the testing equipment used in the drive-tests, and aggregating the information at test level (count, sum, min, max, average, percentiles, etc.). The resulting datasets have a single row per test and hundreds of columns summarizing all the dimensions (i.e., context features like date, time, location, network element information, etc.), technology features (radio, TCP/IP, application, etc.) and the KPI related to that particular test. The data transmitted in the drive tests is synthetic and does not include customer's sensitive information, protected by the European Union General Data Protection Regulation (GDPR) or similar regulations (i.e., the data has been generated by a testing device and not by real users). Finally, potentially sensitive data, such as the identity of the network operator, has been anonymized. The Nokia datasets utilized in this study are rich but not sufficiently large to, e.g., train a neural network. They contain 54 228 and 21 655 rows, respectively, with 1326 columns.

The third dataset we use (Dataset #3) is a MONROE dataset with 120 000 rows and 106 columns. We have obtained access to MONROE and replicated the experiments of the first Nokia dataset (i.e., with 3 MB HTTP file downloads) on a larger scale, although with a different number of features. Moreover, some of the MONROE dataset highlights include the ability to filter by test type, infrastructure, operator, vendor, and transmission technology.

The fourth dataset (Dataset #4) was also obtained by means of the MONROE platform with the main aim of capturing traffic different than TCP. The experiments conducted in this scenario consist in the exchange of data packets from a server we control to a client residing on a MONROE network probe. Packets are sent over QUIC. From a data transport point of view, the novelties introduced by QUIC can be roughly summarized as follows: QUIC offers a combination of TCP reliability and multiplexing capabilities with the speed provided by the UDP transport protocol, over which QUIC is built [46]. In the experiments, we tested multiple possible variants for the congestion control algorithm implemented by QUIC, those being BBR, COPA, New Reno and Cubic. We also conducted two classes of QUIC experiments with sequential and parallel file downloads, respectively. The server can utilize multiple streams to exchange data with the client over QUIC. The number of streams spawned ranged from 1 to 5 across the experiments conducted. To limit the duration of the experiments, each opened connection had a lifespan of 15 seconds, enough to evaluate the behaviour of congestion control algorithms over variable radio conditions. Client nodes located in Norway, Sweden and Spain were utilized to run the experiments. Two types of clients were identified according to their mobility: static and mobile nodes. Among the mobile nodes available in the MONROE platform, the experiments conducted involved those which were installed on Swedish buses. Introducing mobile nodes contributes to increase the heterogeneity of samples with scenarios were the quality of network connectivity can be significantly lower than

in static cases.

The complexity of the QUIC protocol and its use of encryption can hinder the inspection of QUIC traffic to such an extent that a specific format called *qlog* has been defined to organize and collect the information captured during a data exchange [47]. A *qlog* file includes not only QUIC events, but also protocol events, each of which includes a timestamp, the type of event and the value associated as specified in [48]: the timestamp indicates the time at which the event was registered, a field describes the type of event associated to the record, and finally, the data recorded in included. These *qlogs* have been the primary source of information to produce the data that populates the fourth dataset, which emulates the format and features present in the Nokia dataset. Therefore by applying TTrees to QUIC qlogs, we test its flexibility to work with heterogeneous kinds of data without the need of tuning any hyperparameter manually.

Datasets #5 and #6 were specifically designed to evaluate the connectivity to websites that should offer high performance, so that possible network anomalies could be experienced mostly because of radio problems. In this way, we obtain datasets in which radio and transport anomalies are strongly imbalanced. Specifically, Dataset #4 collects measurements run against the Facebook webpage. We obtained the measurements by means of multiple Facebook webpage downloads in various European countries, using MONROE probes. The average downloaded webpage size of Facebook was 72 757.3 bytes and, at each experiment run, we cleaned the cache to make sure that a new complete download occurs. In some countries, such as Sweden and Norway, we had access to mobile nodes mounted on public transport vehicles. Thus, the downloaded page statistics for those countries are a mixture of statistics for static and mobile cellular nodes, hence the slightly higher radio problems which is depicted later in this subsection. As for Italy and Spain, we only report results for static cellular nodes. We run similar measurements with MONROE and Tstat against the Google search engine webpage, which was used to build Dataset #6. In those experiments, We observed an average downloaded size of 17 368.8 bytes, following the same procedure as the Facebook service (repeating downloads, clearing the cache, etc.).

In what follows, we use the first Nokia dataset to validate CIAN/TTrees and showcase its troubleshooting capabilities for the case of HTTP file download operations over LTE networks. We only consider data for a single operator (we anonymize the operator's name for privacy purposes). The other datasets are used to showcase the potentials of CIAN and pros and cons of how the methodology is implemented. In particular, the second Nokia dataset is for HTTP webpage download performed live, over LTE networks, and one single operator as well. The first MONROE dataset Dataset #3 is for HTTP file download over LTE networks, and with one single operator, with tests anonymously conducted from labs and public buses in various European countries; so it is similar to the first Nokia dataset, but with more entries, less features and much more geographical diversity. The MONROE QUIC dataset Dataset #4 is relatively small, with less than 4000 entries, and reports very different families of features with respect to the other datasets. Datasets #5 and #6 are used to show that, in some cases, we need to trade off accuracy for interpretability of CIAN results.

5.3. TTrees in Action and its Validation

For the analysis of the HTTP file download dataset of Nokia, we choose the *throughput* as the desired target KPI. Since there are 6690 samples, the number of categories used for this specific case is $\lfloor(\log_2 6690)/2\rfloor = 6$. Fig. 2 illustrates the results obtained when applying the proportional discretization approach to this KPI in TTrees. The discretized target variable is fed to the CART algorithm to build the knowledge tree. In CART, we fit the knowledge tree using all the 228 features available in the dataset.

Fig. 3 explains a subset of the resulting knowledge tree. For a given sample at the root of the node, the CART algorithm checks if the Abs_RWIN_FirstSec feature (the absolute value of the *TCP received window* during the first second of a connection) is less or equal to the self-learned threshold of about 1.3 MB. If this statement is true, the CART algorithm moves to the left side of the tree. Each node of the knowledge tree reports the value of Gini impurity, i.e., a measure of the fraction of misclassified data points that are reported in the tree leaves that can be reached from that node. It also reports the total number of samples examined in the node (6690 at the root) and the number of samples that corresponds to each discretization category. E.g., $[2185, 1646, 1293, 950, 506, 110]$ in the root node correspond to the number of samples that fall in the "Very bad" (0), "Bad" (1), "OK" (2), "Good" (3), "Very Good" (4), and "Excellent" (5) discretized throughput categories. The class value associated to each node is the one with the highest number of samples ("Very bad", in the example). The full knowledge tree generates a meaningful set of classification rules, each based on a feature, and whose family (radio, TCP window, TCP flags, etc.) is reported

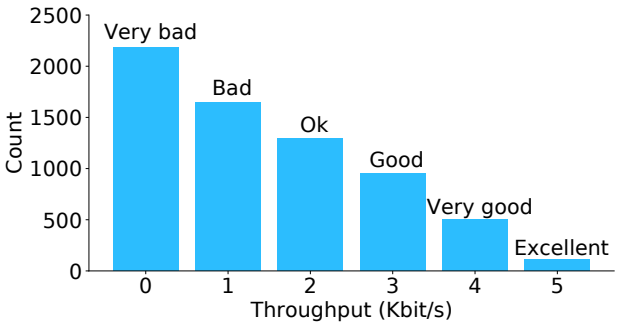


Figure 2: Target KPI (throughput) distribution for Dataset #1 via the proportional discretization approach

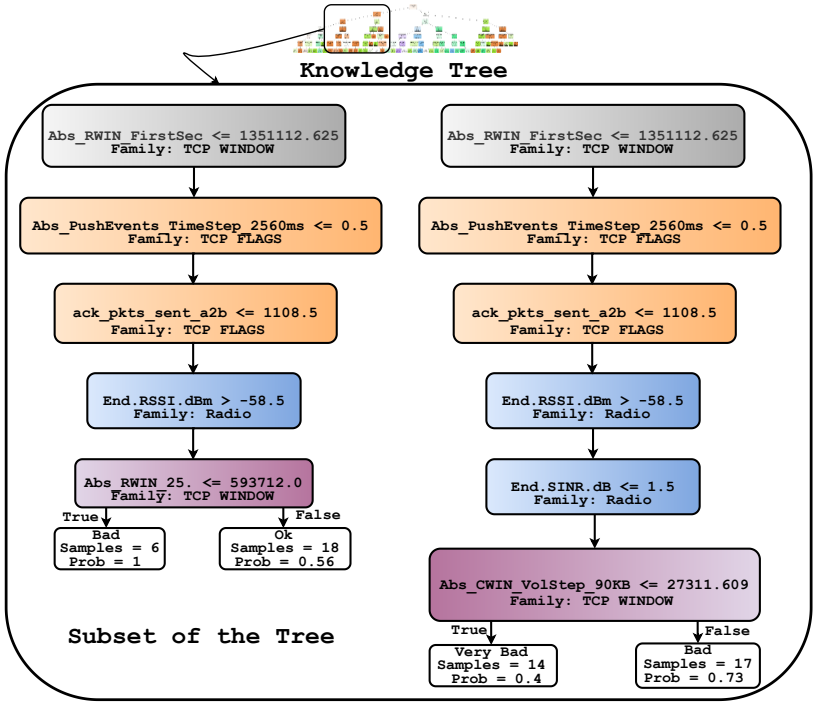


Figure 3: Zoom into a subset of the knowledge tree built for Dataset #1. The figure also reports which family of features the branching variable belongs to (see Table 2). Observe that both branches are the same up to the fourth node, after which the left corresponds to the *true* brach and the right to the *false* branch.

in each node of the tree. That can help expert troubleshooters get an initial idea of what is going on in the cellular network, but since the full knowledge tree is wide, manual inspection and interpretation can be cumbersome.[6]

Fig. 4 shows that the knowledge tree classifies the target samples with high accuracy, although the number of cases in which the predicted class is better than the one observed in the discretization phase is not negligible ('-1' in the figure). TTrees selects these 740 anomalies to continue the analysis.

The number of network aspects that can be analyzed by composing binary clusters in the aspect classifier C_2 with a population of 740 anomalies is $\alpha = \lfloor(\log_2 740)/2\rfloor = 4$. Thus, the number of features to use is $\alpha^2 = 16$. Fig. 5 depicts the 16 most relevant features obtained by using the Mscd metric (left subplot) and by the MI and JMI metrics (right subplot). The figure clearly shows that MI and JMI practically find the same set of features, while Mscd identifies a significantly different subset, which is due to the fact that Mscd accounts for redundancy of features while MI and JMI do not.

In Figs. 6 and 7(a) the redundancy of the binary clusters obtained with the 16 most relevant features is shown,

[6]The full tree is large and cannot be correctly visualized here, so we made it available online at `https://github.com/Mohmoulay/WoWMoM2021`.

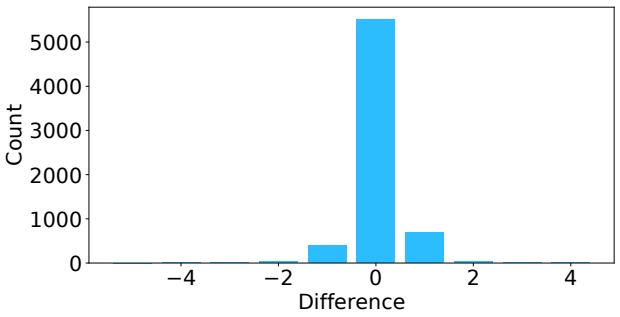
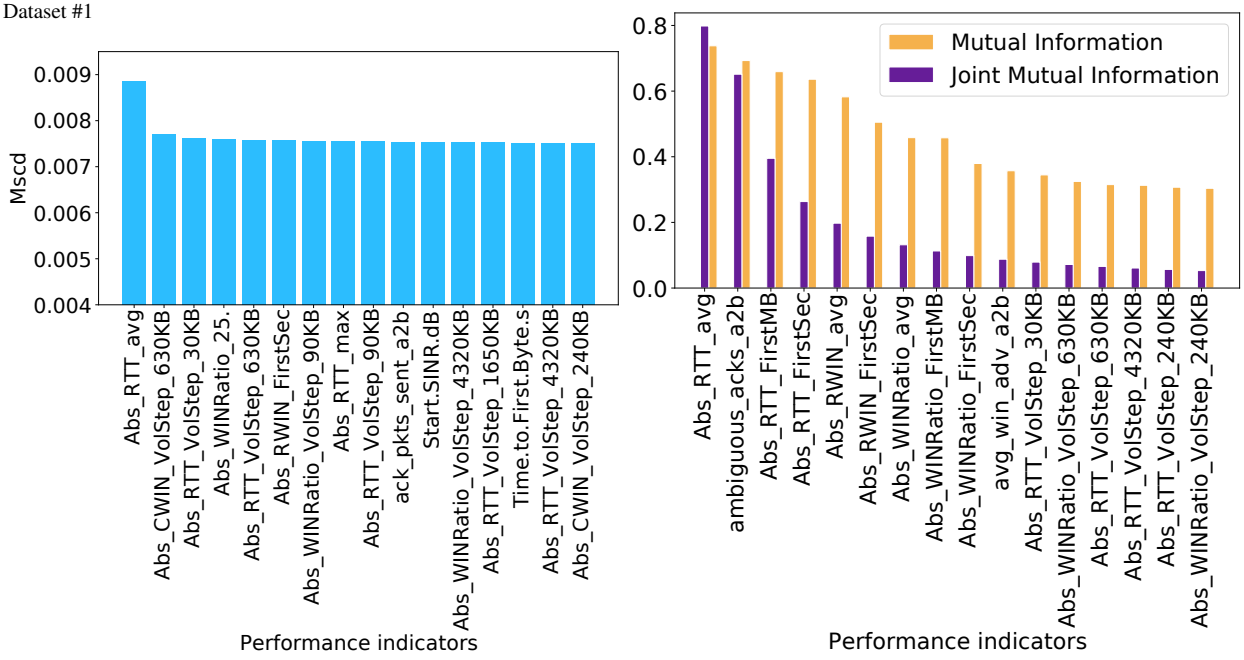


Figure 4: Difference (predicted - discretized) between the class assigned by the knowledge tree and the discretized category for the target KPI of Dataset #1



(a) Miscoding

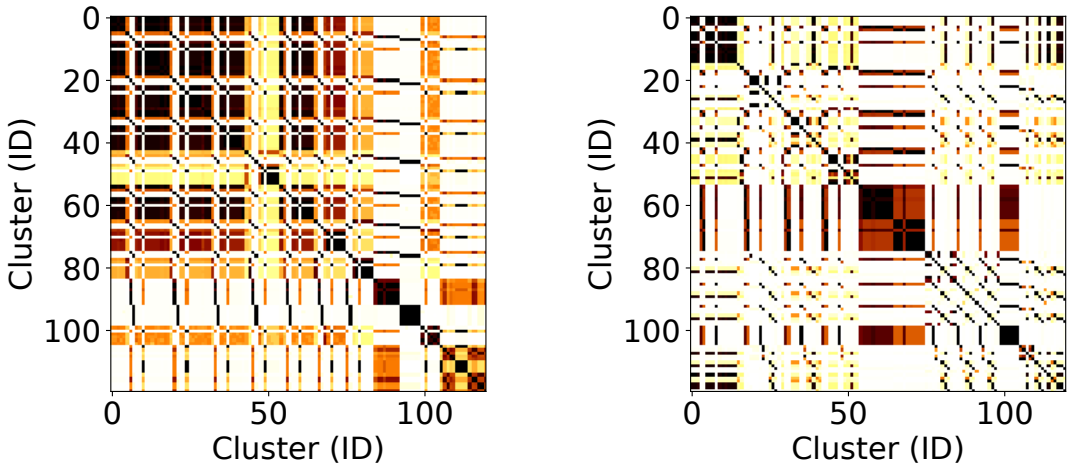
(b) MI and JMI

Figure 5: Ranking of features according to their relevance to the description of anomalies for Dataset #1

for MI, JMI, and Mscd metrics, respectively. The heatmaps shown in the figures represent the NID metric between cluster pairs: the darker the color the lower the distance between cluster pairs (and hence the higher their redundancy). Fig. 7(a) shows lighter colors than the other two heatmaps of Fig. 6, which means that Mscd yields features whose combinations of clusters have lower redundancy. For this reason we will mainly consider the features selected with Mscd in the rest of this section.

Fig. 7(b) illustrates the inertia of the clusters obtained with the features selected with Mscd, and the filtering process based on balancedness and low redundancy criteria. The output of this filtering process is a set of 4 network aspects (in practice, 4 pairs of features) used for binary clustering. The visualization of the filtering results for the selection of relevant aspects obtained with MI and JMI is omitted because it is not relevant for comparison purposes. What matters is instead the set of results on the performance eventually obtained with those metrics, which will be commented later and which are reported in Table 3.

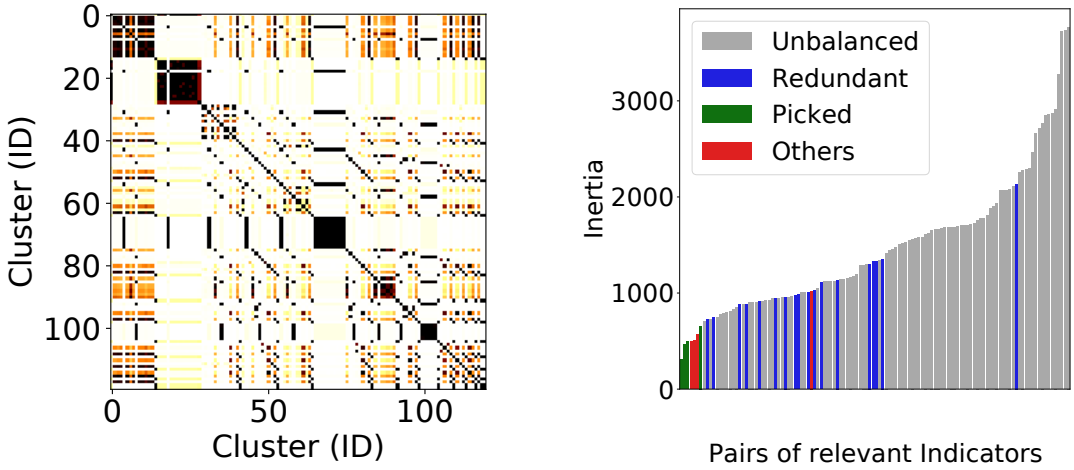
Fig. 8 shows the 4 network aspects obtained with the first Nokia dataset and the corresponding clusters to be used in the aspect classifier C_2. The figure shows that TTrees has identified anomaly clusters characterized by TCP operational parameters, radio parameters, delay parameters and packet anomaly. In STrees, instead, an expert would have to look manually at the "usual suspects" for the presence of anomalies, such as possible radio strength problems, TCP-related problem, packet anomaly and high RTT caused by buffering, and cluster them to see if they yield any



(a) MI-based selection

(b) JMI-based selection

Figure 6: NID heatmap matrix for cluster pairs. Darker colors represent redundancy.



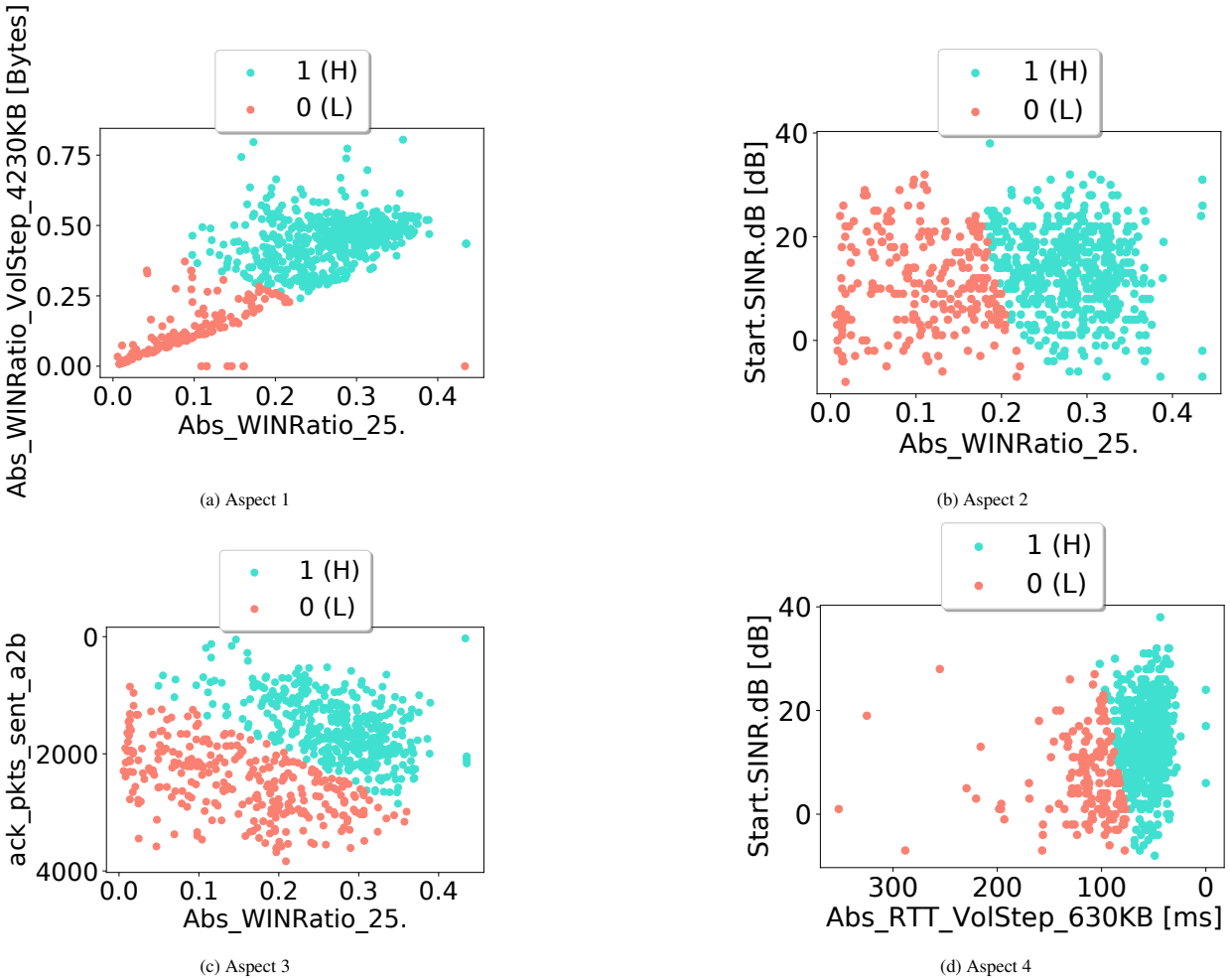
(a) NID of cluster pairs (features selected with Mscd)

(b) Inertia of clusters (combinations of relevant features)

Figure 7: Network aspect selection based on miscoding; lower inertia of k-means clusters is preferred, unless clusters are redundant or imbalanced

groups. In this case, a Nokia cellular expert has identified for us the most relevant features, which led STrees to identify the network aspects visualized in Fig. 9. Aspects identified by TTrees and STrees are quite different, except in both cases they cover the same families of features (radio, TCP window, RTT, packet anomaly). The actual features selected by TTrees could be less intuitive for an expert (e.g., selecting number of acknowledgment instead of packet loss), but without loss in information. In reality, for an expert, Aspect 3 from Fig. 8 makes sense but may not be the first (intuitive) choice; instead, he/she would first look at the packet loss from Fig. 9. Indeed, one of the advantages of the TTrees methodology over STrees is the capability of creating meaningful yet not intuitive inter-family aspects combining radio with TCP, TCP window with packet anomaly, etc.

The corresponding C_2 trees trained with TTrees and STrees are (partially) reported in Fig. 10. The same four families of identifiers are deemed as relevant by supervised and unsupervised approaches: Radio, TCP-based, RTT, Packet anomaly. TTrees detects a dependency in Aspect 4 that is negatively correlated: when the value of the specific feature Abs_RTT_VolStep_630KB (i.e., the RTT observed after averaging over chunks of 630 kB of transmitted data) is higher than 87.909 ms, the throughput is going to be lower. This means that channel capacity will be the limiting factor causing buffering and we have a negatively correlated relation in Aspect 4. Now, if we look at the right side of



(a) Aspect 1

(b) Aspect 2

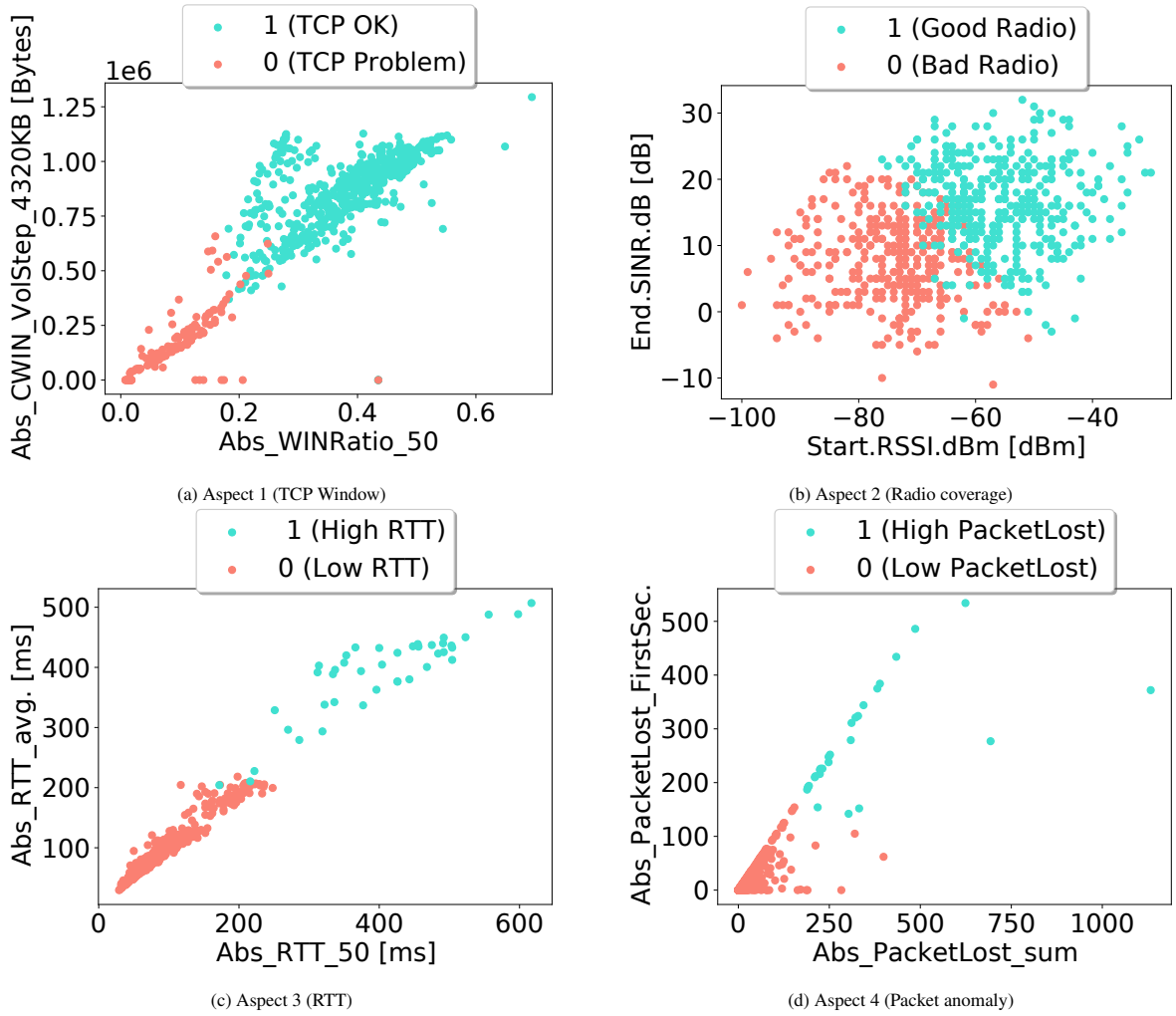
(c) Aspect 3

(d) Aspect 4

Figure 8: Aspects obtained with TTrees for Dataset #1 (using Mscd, see Figs. 5 and 7). The parameters used are Signal-to-interference-plus-noise ratio (SINR), Number of packet acknowledgment sent, absolute TCP Window ratio, and RTT.

Fig. 10, STrees is describing low RTT and high packet loss as well, deriving that result from the same TCP features plus radio. Furthermore, if we look at the whole tree, they both convey similar results. In practice, TTrees and STrees are comparable and show similar classification, taking into account the difference seen in the aspect selected. For instance, the root node is the same: Abs_RTT_avg (i.e., the average RTT observed during file download) is less or equal to 122.432 ms, but the tree leafs can not be identical. Both results are usable by an expert to identify manageable sets of similar tests for troubleshooting. Additionally, the tree rule would allow correct assessments to assign each group to the right troubleshooting department.

The overall accuracy score of C_2 for both approaches is 90%, which means that C_2 can identify 90% of non-anomalies and anomalies alongside their root cause. With the above simple example, we have defined an automated anomaly detection system that is functional to improve the quality of the networking service. Indeed, TTrees can self-identify network performance issues. So it can be used to alert the right support personnel, either the TCP or radio optimization team, in the presented example, which will receive as input the rules from Fig. 10 that raised the alarm. Acknowledging that other classification approaches might yield higher accuracy, we remark that would occur at the expense of interpretability.



(a) Aspect 1 (TCP Window)

(b) Aspect 2 (Radio coverage)

(c) Aspect 3 (RTT)

(d) Aspect 4 (Packet anomaly)

Figure 9: Aspects obtained with STree for Dataset #1

5.4. TCP Performance Evaluation

TTrees has been executed in three different datasets to identify anomalies with TCP data exchanges. The target KPI for Dataset #2 is *Average Session Duration,* discretized into six classes, and for Dataset #3 the KPI is *Throughput,* discretized into eight classes. Since the new datasets are large enough for TTrees, we split them into training and test subsets, with 10% and 20% testing data while applying cross validation 30 times, respectively. Note that, due to the small size of Dataset #4, data could not be split into training and testing sets and thus, cross validation could not be carried out, hence the "NA" entries on the table. Tests are done in classifiers C_1 and C_2 to get the metrics shown in Table 3. The number of anomalous scenarios is 1152 and 6902, with 25 and 36 most relevant features, respectively. TTrees with Mscd was able to identify similar network aspects in Datasets #2 and #3 as it was in Dataset #1: radio, TCP, and RTT. Unfortunately, the selection of the most relevant features did not work well using MI and JMI for Dataset #2 (HTTP livepage DL), since all the pairs of features selected were redundant, and TTrees could not build C_2 (hence the entries "-" in Table 3). This was not surprising to our expert, since the anomalous scenarios of Dataset #2 are very hard to classify, since they contain downloads of webpages with multiple sizes. The fact that Mscd found meaningful network aspects is indeed impressive. For Dataset #1 and #3, the C_2 classifiers show similar classification performance (see Table 3). However, the use of JMI and MI did not produce network aspects easily identifiable by an expert. In summary, using Mscd leads to better performance than the other two metrics, reaching good accuracy scores while producing meaningful sets of rules for troubleshooting.

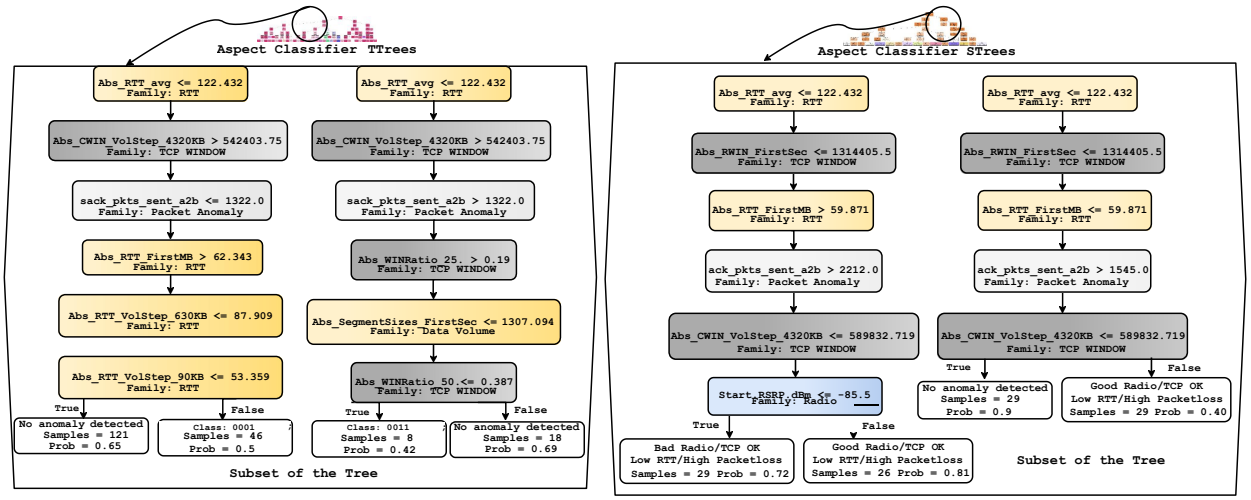


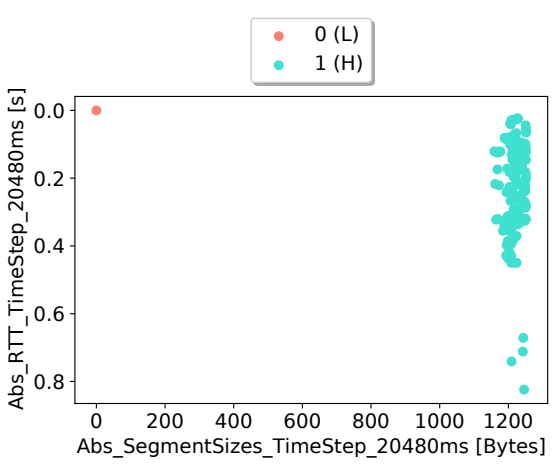
Figure 10: A comparison between the supervised and unsupervised approach showing the resulting aspect classifiers

Table 3: Knowledge (C_1) and Aspect Classification (C_2) trees' performance across the different datasets using multiple aspect selection algorithms (Mutual Information, Joint Mutual Information, and Miscoding)

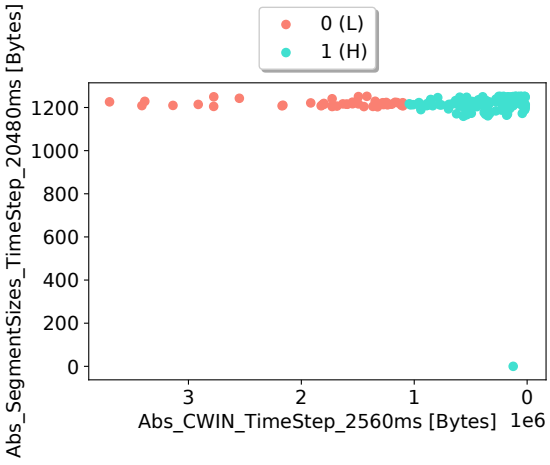
	Classifier	Accuracy	Precision	Recall	F1
Dataset #1 HTTP file DL (Nokia)	C_1	82.5%	NA	NA	NA
	C_2 (MI)	92.0%	NA	NA	NA
	C_2 (JMI)	91.0%	NA	NA	NA
	C_2 (Mscd)	90.0%	NA	NA	NA
Dataset #2 HTTP livepage DL (Nokia)	C_1	82.6%	80.0%	81.5%	80.0
	C_2 (MI)	-	-	-	-
	C_2 (JMI)	-	-	-	-
	C_2 (Mscd)	88.0%	85.0%	86.0%	84.0
Dataset #3 HTTP file DL (MONROE)	C_1	92.0%	90.0%	91.0%	89.0
	C_2 (MI)	95.0%	92.6%	94.4%	92.5
	C_2 (JMI)	95%	94.5%	94.5%	92.7
	C_2 (Mscd)	95.0%	93.0%	94.4%	93.5
Dataset #4 QUIC file DL (MONROE)	C_1	87.3%	NA	NA	NA
	C_2 (MI)	94.0%	NA	NA	NA
	C_2 (JMI)	94.0%	NA	NA	NA
	C_2 (Mscd)	96.0%	NA	NA	NA

5.5. QUIC performance evaluation

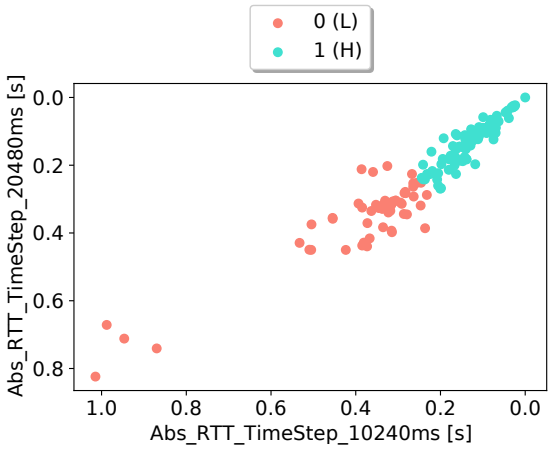
Finally, we use TTrees with our fourth dataset, containing the qlogs of almost 4000 experiments carried out with MONROE. The target KPI for Dataset #4 is the average throughput, which has been discretized into five classes. The congestion control algorithm and execution type used in each experiment were recorded jointly with qlogs and MONROE probe context parameters. Table 4 summarizes per-congestion control algorithm statistics computed to form the knowledge tree C_1 with the initial steps of the TTrees methodology. The table shows that experiments with COPA, Cubic and New Reno systematically yielded low throughput, especially with COPA, while with BBR we can observe a higher throughput, in terms of distribution over the 5 identified classes. The table also shows the number of anomalous cases classified according to network aspects. There were in total eight possible combinations of three network aspects in which to look for anomalies. The three identified network aspects are shown in Fig.11. The



(a) Aspect 1 (Segment sizes)



(b) Aspect 2 (TCP Window)



(c) Aspect 3 (RTT)

Figure 11: Aspects obtained with TTrees for Dataset #4

Table 4: Classification output of TTrees with QUIC experiments – Data grouped per congestion control algorithm (classes with no samples are omitted)

	Knowledge tree C_1		**Aspect classification tree C_2**	
	Throughput class	**Count**	**Anomaly class**	**Count**
BBR	0	363	111	926
	1	304	110	1
	2	243	011	29
	3	87	010	42
	4	1		
COPA	0	983	111	991
	1	14	011	6
Cubic	0	579	111	907
	1	170	011	9
	2	167	010	35
	3	42	000	7
New Reno	0	562	111	928
	1	212	011	16
	2	164	010	15
	3	40	001	3
			000	36

Table 5: Classification output of TTrees with QUIC experiments – Data grouped per experiment execution type (classes with no samples are omitted)

	Knowledge tree C_1		**Aspect classification tree C_2**	
	Throughput class	**Count**	**Anomaly class**	**Count**
Parallel	0	1260	111	1831
	1	413	011	40
	2	208	010	48
	3	70	001	3
			000	29
Sequential	0	1227	111	1921
	1	287	110	1
	2	366	011	20
	3	119	010	44
	4	1	000	14

first two of them show a clear dependency on one parameter only, which is the segment size in the first case and the congestion window in the second case. Note that small segment sizes would naturally lead to low throughput while small congestion window values are typical of bad performers. So, the selection of those network aspects, which was done automatically by TTrees, makes sense. The third network aspect is a combination of RTT statistics taken with different averaging intervals. Indeed, when it comes to evaluate the throughput of congestion-controlled transmissions, it is well known that RTT statistics are a meaningful network aspect to consider as well. In particular, high RTT values tend to indicate the presence of radio limitations.

Table 4 shows that not all combinations of network aspects were observed among the identified anomalies. Of course, the more frequently observed case is that there are no anomalies (class 111), while most of detected anomalies

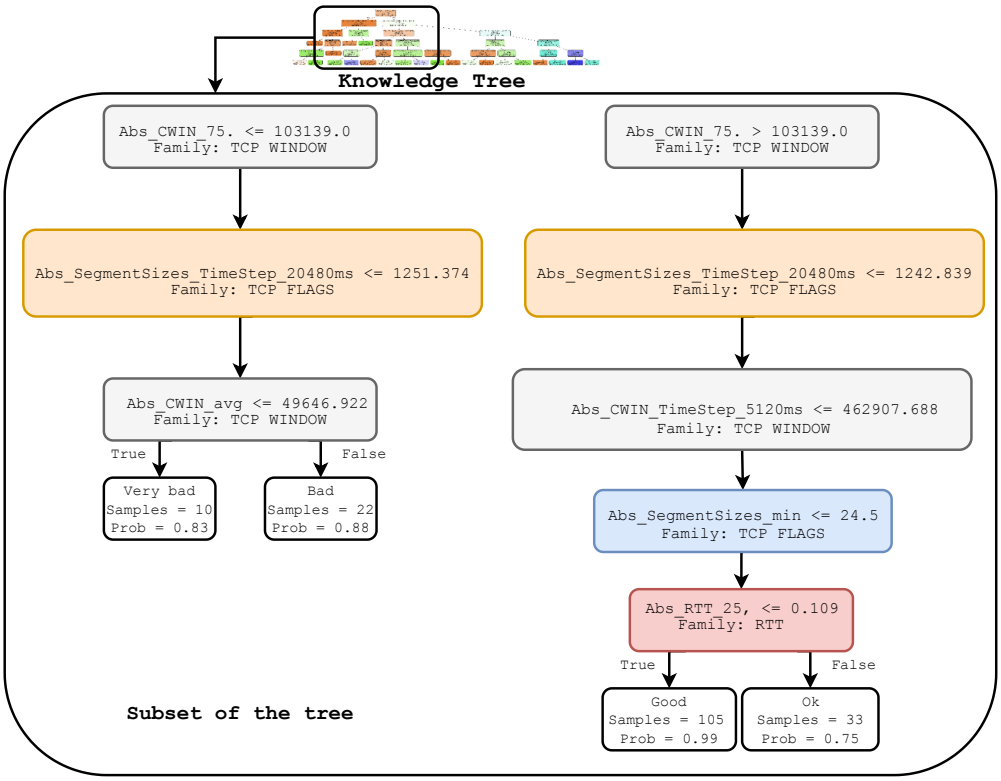


Figure 12: Zoom into a subset of the branches of the knowledge tree C_1 built by TTrees for Dataset #4. The figure also reports which family of features the branching variable belongs to (see Table 2)

show a combination of two or three network aspects. QUIC anomalies with BBR mostly depend on the segment size and RTT, while Cubic and NewReno anomalies also depend on the congestion window. The least number of anomalies was observed with COPA, and are caused by low values of the segment size utilized for transmission. Note that looking at Table 4, we can see that no experiment conducted using BBR falls under the second anomaly class, i.e., the congestion window network aspect does not affect BBR anomalies. This is a meaningful result since with BBR the value of the congestion window (CWIN) is automatically modulated to maintain the RTT within acceptable values, so that CWIN cannot be directly correlated with the experienced throughput.

If we consider parallel and serial QUIC data transmissions separately, Table 5 resumes the distribution of throughput samples—where of course we can see that sequential downloads observe higher per-download throughput—and anomalies. It is interesting to note that anomalies for parallel download experiments show a fundamentally different distribution across anomaly classes with respect to the case of sequential downloads. For instance, while the first network aspect (the segment size) is identified as a common cause of anomaly in all cases, the second network aspect (i.e., the congestion window size) is more relevant for parallel downloads. This result is meaningful because segment size always affects transmission rate, while the congestion window is affected by the number of ongoing transmission flows.

Note that, due to the small size of Dataset #4, data could not be split into training and testing sets and thus, cross validation could not be carried out. The knowledge tree, shown in Fig. 12, yielded 199 anomalous scenarios in total. As mention before, here we selected only three network aspects, which were extracted from a set of nine most relevant features. The aspect selection of TTrees automatically deemed appropriate to stop the search of network aspects to three, because adding a fourth one would have been redundant. We remark that this was done automatically, and we did not have to manually change the number of network aspects to select. The accuracy of the aspect classification tree for the QUIC dataset of Fig. 13, was the highest of all C_2 trees, with a value of 96% and includes the three types of anomalies identified. Also for the case of QUIC, using miscoding rather than (join) mutual information for aspect

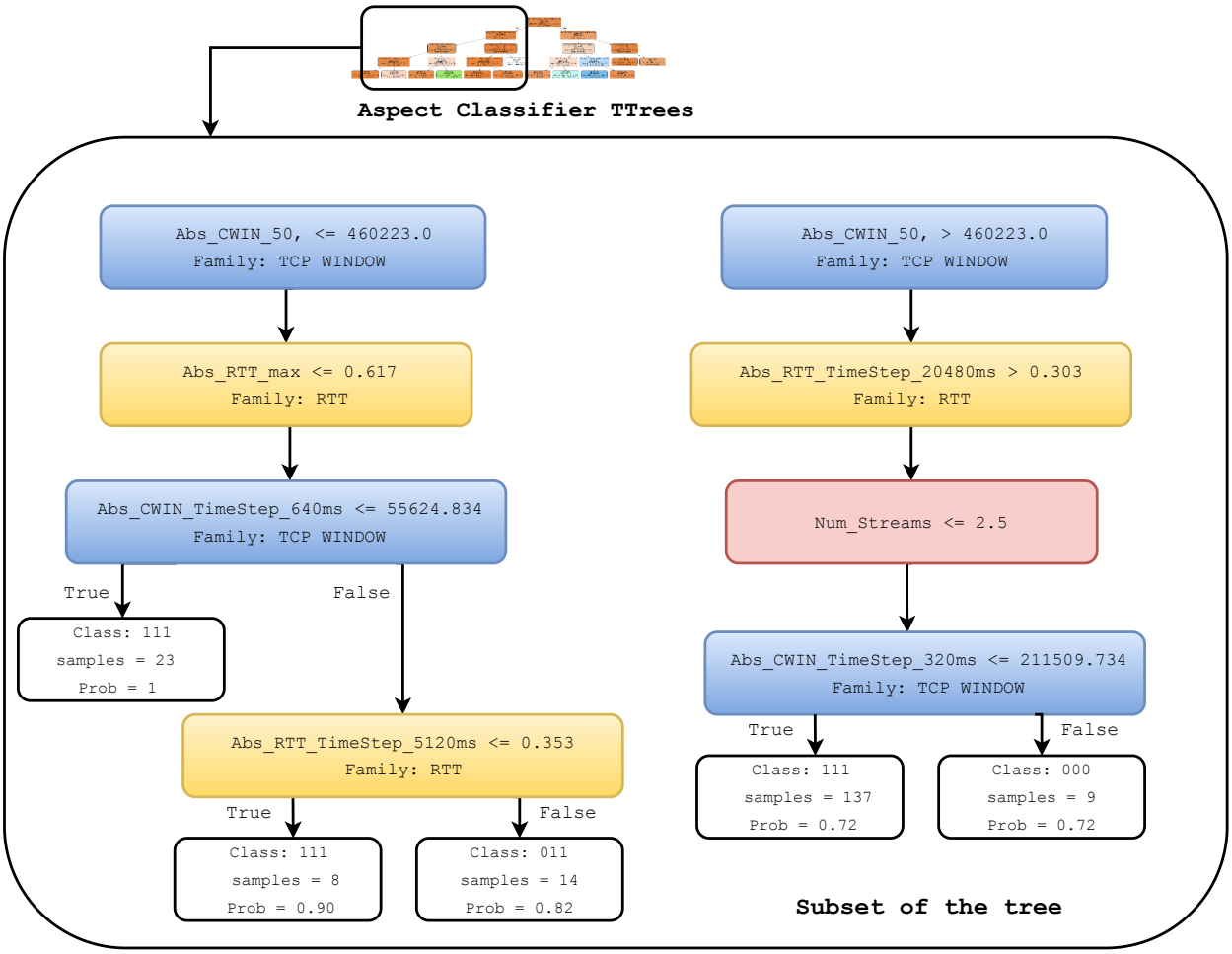


Figure 13: Zoom into a subset of the branches of the aspect classification tree C_2 built by TTrees for Dataset #4. The figure also reports which family of features the branching variable belongs to (see Table 2)

selection was eventually advantageous in terms of accuracy, as visible in Table 3.

With Dataset #4, trees C_1 and C_2 are smaller than with the other datasets, which allows us to analyze them more in detail. A general analysis of the classification trees C_1 (partially depicted in Fig. 12) and C_2 (Fig. 13) shows that critical QUIC features were properly selected to classify the performance of tests. In particular congestion window size (specifically, its 75th percentile) is at the root node of C_1. Indeed, given the characteristics of the tests performed using different congestion algorithms and assuming similar radio conditions, it is clear that the main features differentiating each type of test has to be the congestion window related statistics. Other features like the number of streams, buffering/delay (RTT), packet loss or segment sizes are also correctly classifying each test. For example, RTT values above 100 ms identify lower throughput samples due to capacity limitations (high buffering). A higher number of streams is also normally identifying better samples in the knowledge tree C_1 built by TTrees for Dataset #4.

The aspect classification tree C_2 tells that, with high probability, we either have an anomaly due to a low congestion window (maybe due to packet loss), or a combination of the three identified network aspects. For the first case we have two leaves of the tree, and in both leaves we can see that CWIN and RTT in the initial five seconds were OK, but at the end of the test they were not. This might be due to packet loss. In the case of combined anomalies, the tree shows that there was low radio capacity or the congestion algorithm was too aggressive. In both cases we have identified the cause of anomalous performance of QUIC with the same methodology used to identify anomalous performance of TCP.

Table 6: A summary of the knowledge tree's (C_1) performance in TTrees utilizing Facebook & Google as a service in all the countries part of the MONROE project with all their operators.

Services	Country	Operators	Accuracy (%)	# Samples	# Anomalous samples
Facebook (Dataset #5)	**Sweden**	$op0$_sw	96.0	183240	3162
		$op2$_sw	96.6	235568	3346
	Norway	$op0$_no	96.5	160290	2235
		$op1$_no	96.6	130334	2139
		$op2$_no	95.4	128804	2688
	Italy	$op0$_it	96.9	66420	1008
		$op1$_it	96.5	83966	1049
	Spain	$op0$_es	97.5	26795	320
		$op1$_es	95.5	30575	380
		$op2$_es	96.3	86450	1044
Google (Dataset #6)	**Sweden**	$op0$_sw	92.7	146327	3061
		$op2$_sw	98.9	180784	953
	Norway	$op1$_no	99.2	127628	469
	Italy	$op0$_it	96.8	66420	1008
		$op1$_it	96.5	83966	1049
	Spain	$op0$_es	98.4	24530	117
		$op1$_es	91.4	2078	32718
		$op2$_es	88.5	77729	7145

5.6. Modifications needed to deal with imbalancedness of datasets

To further extend the evaluation of TTrees, we employ Datasets #5 and #6, collected with MONROE. These datasets are collected from real operational mobile networks over which we do not have control. However, we have tested the connectivity to Facebook and Google webpages, which are highly replicated across the network, and easily reachable without having to traverse large portions of the cellular and core networks of the mobile operators to which MONROE probes were connected. As such, these datasets suffer imbalancedness, meaning that TCP problems are rare, while radio conditions might impair performance with much higher frequency, especially for mobile MONROE probes. Therefore, the analysis of Datasets #5 and #6 is suitable to showcase possible drawbacks due to the use of simple ML mechanisms in TTrees, and in particular the clustering operated with *k*-means, which is known to suffer in case of parsing imbalanced datasets [49, 50].

Table 6 summarizes the results obtained with TTrees in terms of accuracy and the number of samples concerning the knowledge tree for Datasets #5 and #6. The table shows results classified according to the origin of the data collected, i.e., per country of test and per operator. Similarly, Table 7 showcases the performance of TTrees when it comes to the aspect classification tree for the same datasets. For both datasets, the accuracy obtained with TTrees in both trees C_1 and C_2 is high, and the number of anomalies is as limited as it could be expected. However, Table 8 reveals that TTrees does not always manage to reach a fully meaningful classification of anomaly causes. In particular, the fourth and fifth columns of Table 8 show that the number of aspects identified with TTrees, and of the corresponding families, by means of *k*-means clustering is small for these datasets. This is a problem because the less aspects and families are extracted by the algorithm, the poorer is the precision with which the cause of an anomaly can be explained. Moreover, Table 8 does not report the results for some of the tested networks because in those cases there were not enough aspects to generate C_2. The problem with the analysis of these datasets is that the anomalies found by means of C_1 are strongly imbalanced. For instance, Figure 15 shows that *k*-means founds most of the anomalies of C_1 concentrated in one of the two clusters extracted after applying binary classification using the most relevant features. In some scenarios, such as the ones displayed in those figures, this led to the impossibility of finding a set

Table 7: A summary of the aspect classification tree's (C_2) performance with TTrees utilizing Facebook & Google as a service in all the countries part of the MONROE project with all their operators.

Services	Country	Operators	Accuracy (%)	# Samples	# Anomalous samples
Facebook (Dataset #5)	**Sweden**	$op0$_sw	98.5	183240	104
		$op2$_sw	98.9	235568	295
	Norway	$op0$_no	98.9	160290	2235
		$op1$_no	98.6	130334	2139
		$op2$_no	98.8	128804	770
	Italy	$op1$_it	98.7	83966	5
	Spain	$op0$_es	99.0	26795	18
		$op1$_es	98.9	30575	38
		$op2$_es	99.0	86450	1044
Google (Dataset #6)	**Sweden**	$op0$_sw	98.5	146327	105
		$op2$_sw	98.9	180784	322
	Norway	$op1$_no	99.6	127628	23
	Italy	$op0$_it	98.9	66420	89
		$op1$_it	99.0	83966	0
	Spain	$op0$_es	99.5	24530	0
		$op2$_es	95.5	77729	350

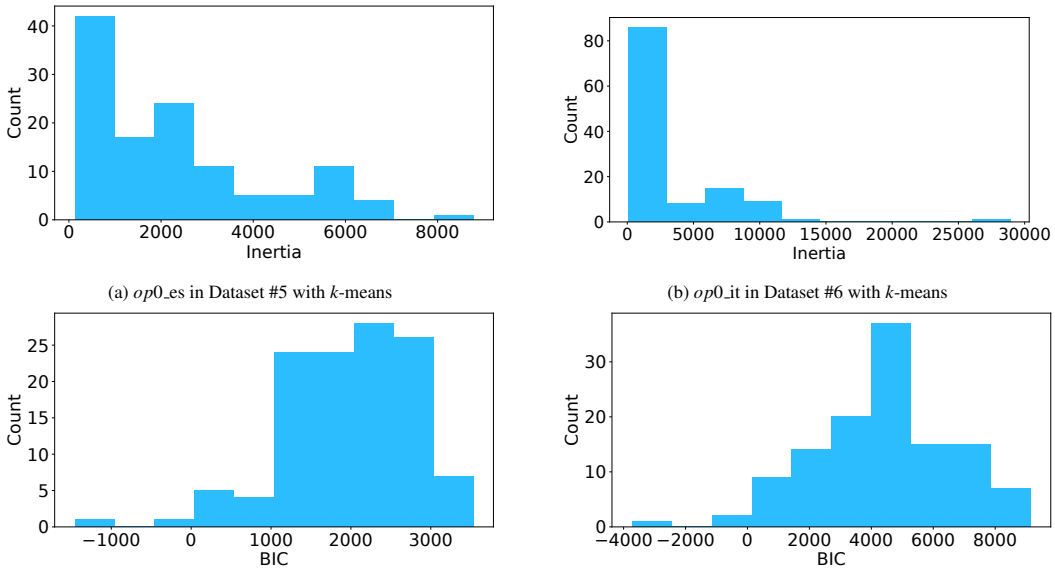
of features that fell under the balancedness threshold, and, thus, no aspects were obtained for labelling anomalous classes in C_2. Therefore, applying k-means on these imbalanced C_1 anomalies can incur performance problems, as actually witnessed in our experiments.

Note that we have designed our CIAN methodology in a modular way, and the TTrees implementation relaying on k-means is only a possible implementation with some strong advantages (it relies on simple algorithms and is interpretable) and some cons due to the limitations of the selected algorithms. However, in CIAN, methodological phases are well separated and the overall system is highly modular. Therefore, tools and algorithm used for a particular implementation can be easily exchanged guaranteeing no inter-dependency between technologies at different steps. Considering this, we can think of combating imbalancedness through a relatively simple modification of TTrees to implement CIAN. The alternative implementation of the methodology can simply differ from TTrees in how problems are clustered. In particular, we tested a modified version of TTrees obtained by interchanging k-means with Gaussian Mixture Model (GMM) [51], which is robust to imbalancedness and operates clustering in a very flexible way, although with less interpretable results.

5.6.1. Modified TTrees with GMM clustering

It is important to bear in mind that GMM is a model-based clustering approach that works with the concept of soft clustering, meaning that data is not just assigned to a cluster, but instead it has a likelihood of belonging to it [51]. Therefore, the objective of applying GMM to implement CIAN is to maximize this probability, a concept which can be directly associated to that of cluster tightness measured through inertia. Indeed, the previously defined TTrees implementation computes the inertia score for each pair of attributes. Instead, GMM requires to use of a different metric to draw performance comparisons between different pairs of attributes. The most well-known score for the selection of the optimal model in model-based clustering is the Bayesian Information Criterion (BIC) [52], so we use the BIC metric. Like inertia, the BIC metric is a penalty measure, which has to be minimized.

To show that GMM and k-means behave in a substantially different way, we evaluate the behavior of BIC and inertia obtained by applying GMM and k-means to form C_2 with the data relative to two operators within what collected in Dataset #5. Specifically, Figure 14 depicts the clustering performance values for the pairs of attributes extracted for Dataset #5 with $op0$ in Spain, using GMM or k-means, respectively. Given that having lower inertia and



(a) *op*0_es in Dataset #5 with *k*-means

(b) *op*0_it in Dataset #6 with *k*-means

(c) *op*0_es in Dataset #5 with GMM

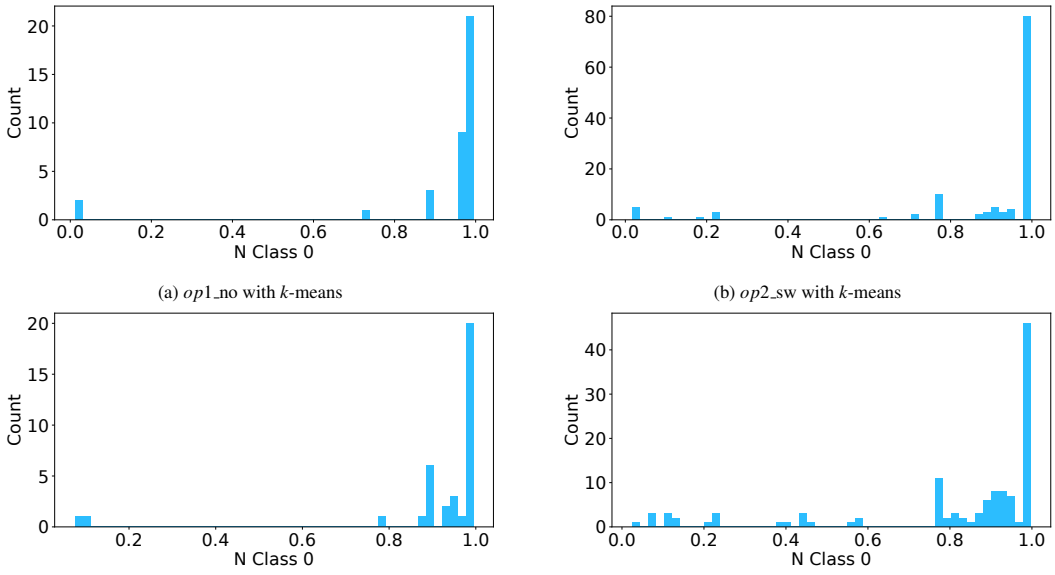
(d) *op*0_it in Dataset #6 with GMM

Figure 14: Histogram of Inertia (a, b) and BIC (c, d) values extracted during aspect clustering for all relevant aspects and with two operators in Spain and Italy. The lower the BIC or inertia the cleaner cluster grouping and splitting of samples is overall

BIC values translates into better separation between clusters, we see that the BIC metric reports values that are clearly lower than the ones extracted with inertia, and, more importantly, grouped within a considerably smaller range. This difference in the consistency of clustering performance may be an indicative of unstable behaviour of *k*-means, thus showing the presence of imbalancedness in the dataset. Moreover, according to the analysis conducted, low inertia would correspond to values between a 40 and 100, which as it is shown in the image, it is not the case for most attributes. This translates in the obtained clusters not being the most optimal, highlighting an under-performance of *k*-means compared to previous datasets and resulting of this imbalancedness. Figure 14 shows that the same analysis carried out for another operator and dataset (in Italy and with Dataset #6) yields an upper threshold for inertia values higher than the one obtained with BIC and a similar tendency in the distribution of data compared with the scenario mentioned previously. This proves that the imbalancedness of the dataset is consistent and observable across multiple operators and scenarios and thus it is a direct result of the nature of the experiments conducted.

Thus, using BIC, GMM is able to identify substantially different behaviors across the data collected for one operator or the other, while *k*-means might not be able to do so, at least when the datasets are imbalanced. Figure 15 clearly shows that the anomalies obtained from C_1 are indeed imbalanced for the two cases analyzed in Figure 14. During the clustering phase of the methodology, samples classified as anomalous are fit into one of two clusters for each aspect: a cluster of problematic samples and one of compliant (non-problematic) samples. Since "N Class 0", which is the ratio of samples falling under the non-problematic class for this binary classification problem, only takes values that are very close to 0 or 1, the anomalies are indeed imbalanced, as opposed to being evenly distributed.

By comparing aspect classification performance across more scenarios, the sixth and seventh columns of Table 8 show that using GMM instead of *k*-means, with Datasets #5 and #6 finds more aspects. The last two column show that GMM finds most of the aspects found by *k*-means and identifies additional ones. This shows that GMM tends to be more general than *k*-means. However, the results obtained with *k*-means are more interpretable. Interpretability in decision trees is tightly correlated to the tree's size. Thus, the higher the number of nodes, the less interpretable the tree is. The significant difference in the number of aspects identified with GMM translates in an increase in the depth and branching of C_2. Therefore, although GMM was able to compute a solution in those particular cases, it yielded clusters and an aspect classification trees harder to traverse and understand. This relates to the fact that although GMM can handle complex patterns in data, it does so at the expense of interpretability [53].



(a) $op1$_no with k-means

(b) $op2$_sw with k-means

(c) $op1$_no with GMM

(d) $op2$_sw with GMM

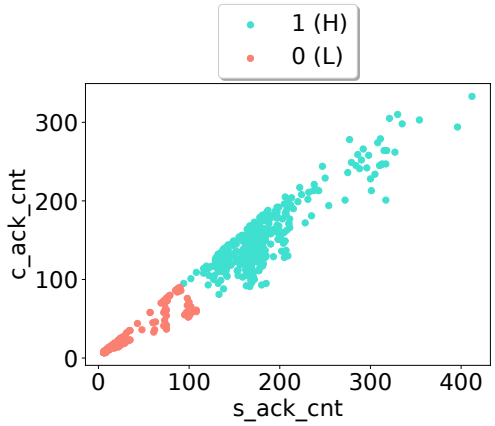
Figure 15: Histogram of the ratio of samples falling under the non-problematic class during aspect clustering for all relevant aspects in Dataset #5 and with two operators in Sweden and Norway. As it can be seen, most aspects, especially with k-means, yield imbalanced clusters of anomalies.

5.6.2. Take away message

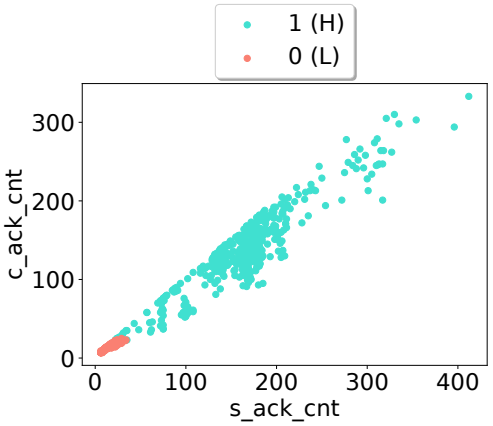
In case the dataset is balanced k-means and GMM convey similar clustering each grouping the clusters based on how the algorithm works underneath as shown in Figures 16 and 17. This is true for Datasets #1 to #4. However, Table 8 tells that the differences in the aspect families extracted from the application of k-means and GMM are significant in Datasets #5 and #6. The table reports the number of intersecting aspects and the ratio between intersecting aspects and the size of the smallest aspect set among the ones identified by k-means and GMM. The value of the ratio is, in most cases, close to one, which hints to the fact that the intersection of aspects found with the two approaches is similar to the smallest set identified by either GMM or k-means. Moreover, it is significant that in all the subsets available in Datasets #5 and #6, GMM finds a group of attributes equal or larger to the one extracted with k-means. In practice, as shown in the table, all the aspect families detected by k-means are extended by GMM, which is indicative of the fact that k-means identifies a subset of the data extracted with GMM. Thus, differently from GMM, k-means enforces a more aggressive filtering behaviour when selecting aspects, and this behavior is strongly affected by the balancedness filter used in the proposed methodology.

This difference between using GMM or k-means is clearly displayed, e.g., in Figure 15, in which we can see how GMM is able to identify a significantly higher number of aspects closer to an ideally balanced distribution of samples. It is also important to note that in some cases, k-means does not find a sufficient number of aspects and is not capable of identifying the root for the existence of anomalies. In fact, when TTrees with k-means does not obtain an aspect label, no aspect families are highlighted in C_2, and TTrees cannot identify clearly whether a group of anomalies as classified by the decision tree C_1 is problematic or not for the target KPI. Indeed, the fact that some operators are not included in Table 8 alludes to the failure of some experiments utilizing k-means. We observed that TTrees with the latter clustering algorithm was not able to generate C_2 due to a highly imbalanced number of anomalies obtained from C_1. On the contrary, the application of GMM as the clustering algorithm in the same scenarios always resulted in a complete aspect classification tree. Note also that, although GMM was able to compute a solution in those particular cases, it yielded clusters and an aspect classification trees harder to traverse and understand. Therefore, at the end of the day, if imbalanced anomalies are encountered, a feasible approach would be to switch the clustering algorithm to one that is able to handle this particular data distribution, thus, allowing for a higher flexibility.

Ideally, we want an easy-to-interpret separation to distinguish between anomaly samples, and this is why in TTrees we have opted for k-means. However, in case of imbalanced datasets, a more complex separation of anomalies into

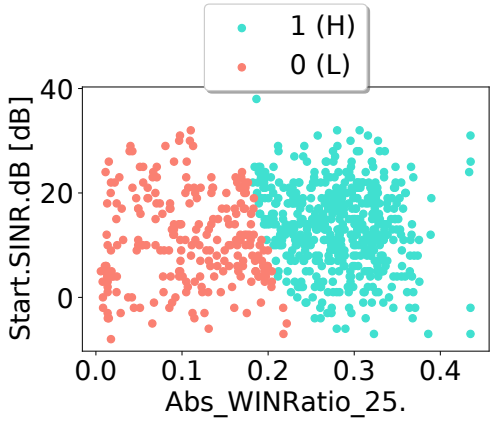


(a) Aspect 1 (Packet anomalies) with *k*-means

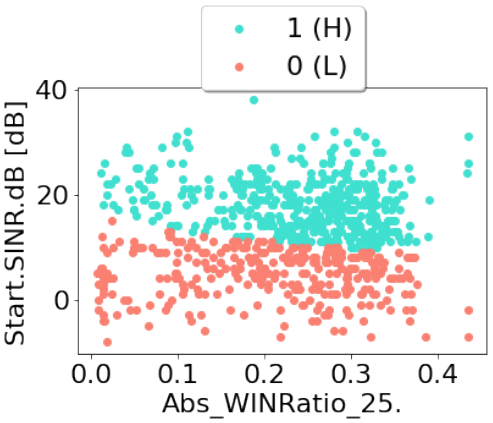


(b) Aspect 1 (Packet anomalies) with GMM

Figure 16: Cluster grouping using *k*-means and GMM as the cluster algorithm using Google Dataset and *op0_it*



(a) Aspect 1 (TCP Congestion Window Vs Signal-To-Noise-plus-Interference Ratio (SINR)) with *k*-means



(b) Aspect 1 (TCP Congestion Window Vs Signal-To-Noise-plus-Interference Ratio (SINR)) with GMM

Figure 17: Cluster grouping Comparsion using *k*-means and GMM as the cluster algorithm for Dataset #1

Table 8: A comparison of the number of aspects and aspect families extracted from C_2 between k-means and GMM for the Google & Facebook datasets in all the countries part of the MONROE project. The table also shows the number of aspects that are common to both algorithms as well as the ratio, calculated from the division of the intersection by the size of the smallest set of aspects between k-means and GMM.

Service	Country	Operators	# k-means Aspects	k-means Aspect families	# GMM Aspects	GMM Aspect families	# Aspect intersection	Ratio
Facebook (Dataset #5)	**Sweden**	$op0$_sw	6	RTT, TCP Window, Packet anomaly	8	RTT, TCP, TCP Window, Packet anomaly	4	0.667
		$op2$_sw	4	RTT, Packet anomaly	10	RTT, TCP, TCP Window, Packet anomaly	4	1.0
	Norway	$op0$_no	2	-	11	RTT, TCP, TCP Window, Packet anomaly	2	1.0
		$op1$_no	2	-	10	RTT, TCP, TCP Window, Packet anomaly	1	0.5
		$op2$_no	5	RTT, TCP, TCP Window	10	RTT, TCP Window, Packet anomaly	4	0.8
	Italy	$op1$_it	8	RTT, TCP, TCP Window, Packet anomaly	10	RTT, TCP, TCP Window, Packet anomaly	4	0.5
	Spain	$op0$_es	4	RTT, TCP	7	RTT, TCP, TCP Window	3	0.75
		$op1$_es	6	RTT, TCP	7	RTT, TCP, TCP Window	5	0.833
		$op2$_es	2	-	7	TCP, TCP Window, Packet Anomaly	2	1.0
Google (Dataset #6)	**Sweden**	$op0$_sw	10	RTT, TCP, TCP Window	10	RTT, TCP, TCP Window, Packet anomaly	4	0.4
		$op2$_sw	8	RTT, TCP, TCP Window	9	RTT, TCP, TCP Window	7	0.875
	Norway	$op1$_no	5	RTT, TCP Window, Packet anomaly	9	RTT, TCP Window, Packet anomaly	5	1.0
	Italy	$op0$_it	9	RTT, TCP, TCP Window	8	RTT, TCP, TCP Window	7	0.875
		$op1$_it	10	RTT, TCP, Packet Anomaly	12	RTT, TCP, Packet Anomaly	8	0.8
	Spain	$op0$_es	1	-	1	-	1	1.0

clusters is needed, and can be achieved by using GMM, at the expenses of interpretability. Therefore, at the end of the day, if anomalies are balanced, k-means is a suitable tool to be used to implement CIAN, whereas, should imbalanced anomalies be encountered, a feasible approach would be to switch to GMM. In practice, we could extend TTrees by automatically checking the presence of imbalanced anomalies, and then automatically select the appropriate clustering algorithm between k-means and GMM, or another one that is able to handle the particular data/anomaly distribution observed in the dataset. This would allow for a higher flexibility in the implementation of CIAN.

6. Conclusions

In this article, we have developed, validated and applied a methodology to identify the root causes of network anomalies by means of interpretable ML algorithms. Our methodology, named CIAN, allows for automatic identification of the networking aspects that cause anomalous behaviors. Such anomalies cannot be immediately and directly explained by observing the network features. In fact, the potentially large number of network aspects, even in the presence of a limited number of features and samples per feature, makes it impossible to manually inspect and correlate. With CIAN, we have purposely developed a radically novel methodology and implemented it with Python. Specifically, we have leveraged the key observation that if a network behavior cannot be modeled (or learned by ML), it is a symptom of network anomaly. We therefore easily identify performance anomalies, after which we are able to use unsupervised ML and Kolmogorov complexity-inspired tools to smartly search for the aspects that are more relevant to explain where the anomaly is rooted. In particular we have introduced a novel metric, miscoding, for the evaluation of redundancy and relevance of features. The final outcome of our proposed methodology is a set of easy-to-interpret classification rules, that, in case of performance anomaly, allow for automatically alerting the appropriate departments for corrective actions. To do so, CIAN only needs little volumes of samples for the features. CIAN is not specifically designed for a network protocol or service or dataset type, as we have validated in this work by applying its

implementation, TTrees, to the analysis of TCP and QUIC with data gathered with different granularity, formats and richness of experimentally collected metrics. Indeed, TTrees allows to fully automatize network troubleshooting with high accuracy and fast training, as shown in this article with the help of real data gathered from operational cellular networks in various countries. We have also evaluated the impact of imbalanced anomalies on CIAN and verified that TTrees might suffer performance issues in that case, which can however be obviated by simply re-placing (possibly by means of an automated software selection process) k-means by GMM in the clustering of anomaly aspects. This highlights the flexibility and modularity of the CIAN methodology.

Acknowledgments

The work was partially supported by the Regional Government of Madrid (CM), under grant EdgeData-CM (P2018/TCS4499, cofunded by FSE & FEDER). The work was also partially supported by the Spanish State Research Agency (AEI) PID2019-109805RB-I00/ AEI/10.13039/501100011033 grant (ECID project).

References

[1] M. Moulay, R. G. Leiva, V. Mancuso, P. J. Rojo Maroni, A. Fernández Anta, Ttrees: Automated classification of causes of network anomalies with little data, in: 2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2021, pp. 199–208. doi:10.1109/WoWMoM51794.2021.00033.

[2] P. Yang, Y. Xiao, M. Xiao, S. Li, 6G wireless communications: Vision and potential techniques, IEEE Network 33 (4) (2019) 70–75. doi:10.1109/MNET.2019.1800418.

[3] J. P. Santos, R. Alheiro, L. Andrade, A. L. Valdivieso-Caraguay, L. I. Barona-López, M. A. Sotelo-Monge, L. J. Garcia-Villalba, W. Jiang, H. Schotten, J. M. Alcaraz-Calero, Q. Wang, M. J. Barros, SELFNET framework self-healing capabilities for 5G mobile networks, Trans. Emerg. Telecommun. Technol. 27 (9) (2016) 1225–1232. doi:10.1002/ett.3049.

[4] A. Asghar, H. Farooq, A. Imran, Self-healing in emerging cellular networks: Review, challenges, and research directions, IEEE Communications Surveys & Tutorials 20 (3) (2018) 1682–1709.

[5] O. Loyola-González, Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view, IEEE Access 7 (2019) 154096–154113.

[6] A. Holzinger, M. Plass, K. Holzinger, G. Crisan, C. Pintea, V. Palade, A glass-box interactive machine learning approach for solving NP-hard problems with the human-in-the-loop, arXiv preprint arXiv:1708.01104 (2017).

[7] S. Dhanorkar, C. T. Wolf, K. Qian, A. Xu, L. Popa, Y. Li, Who Needs to Know What, When?: Broadening the Explainable AI (XAI) Design Space by Looking at Explanations Across the AI Lifecycle, in: Designing Interactive Systems Conference 2021, DIS '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 1591–1602. doi:10.1145/3461778.3462131.
URL https://doi.org/10.1145/3461778.3462131

[8] F. K. Došilović, M. Brčić, N. Hlupić, Explainable artificial intelligence: A survey, in: MIPRO, 2018.

[9] M. Langer, D. Oster, T. Speith, H. Hermanns, L. Kästner, E. Schmidt, A. Sesing, K. Baum, What do we want from Explainable Artificial Intelligence (XAI)? – A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research, Artificial Intelligence 296 (2021) 103473. doi:https://doi.org/10.1016/j.artint.2021.103473.
URL https://www.sciencedirect.com/science/article/pii/S0004370221000242

[10] T. Ogata, A. Takeuchi, S. Fukuda, T. Yamada, T. Ochi, K. Inoue, J. Ota, Characteristics of skilled and unskilled system engineers in troubleshooting for network systems, IEEE Access 8 (2020) 80779–80791.

[11] N. Bostrom, E. Yudkowsky, The ethics of artificial intelligence, The Cambridge handbook of artificial intelligence 1 (2014) 316–334.

[12] O. Alay, A. Lutu, M. Peón-Quirós, V. Mancuso, T. Hirsch, K. Evensen, A. Hansen, S. Alfredsson, J. Karlsson, A. Brunstrom, A. Safari Khatouni, M. Mellia, M. A. Marsan, Experience: An open platform for experimentation with commercial mobile broadband networks, in: ACM MobiCom, 2017. doi:10.1145/3117811.3117812.

[13] 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, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[14] T. Marwala, Causality, Correlation and Artificial Intelligence for Rational Decision Making, WORLD SCIENTIFIC, 2015. arXiv:https://www.worldscientific.com/doi/pdf/10.1142/9356, doi:10.1142/9356.
URL https://www.worldscientific.com/doi/abs/10.1142/9356

[15] B. Huang, K. Zhang, J. Zhang, J. Ramsey, R. Sanchez-Romero, C. Glymour, B. Schölkopf, Causal discovery from heterogeneous/nonstationary data with independent changes (2019). doi:10.48550/ARXIV.1903.01672.
URL https://arxiv.org/abs/1903.01672

[16] A. McGovern, R. Lagerquist, D. J. Gagne, G. E. Jergensen, K. L. Elmore, C. R. Homeyer, T. Smith, Making the black box more transparent: Understanding the physical implications of machine learning, Bulletin of the American Meteorological Society 100 (11) (2019) 2175 – 2199. doi:10.1175/BAMS-D-18-0195.1.
URL https://journals.ametsoc.org/view/journals/bams/100/11/bams-d-18-0195.1.xml

[17] J. Pearl, The seven tools of causal inference, with reflections on machine learning, Commun. ACM 62 (3) (2019) 54–60. doi:10.1145/3241036.
URL https://doi.org/10.1145/3241036

[18] D. Garreau, U. von Luxburg, Explaining the explainer: A first theoretical analysis of lime, in: Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS), Vol. 108 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 1287–1296.
URL http://proceedings.mlr.press/v108/garreau20a.html
[19] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 4768–4777.
[20] A. Shrikumar, P. Greenside, A. Kundaje, Learning important features through propagating activation differences, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, p. 3145–3153.
[21] A. Binder, G. Montavon, S. Bach, K. Müller, W. Samek, Layer-wise relevance propagation for neural networks with local renormalization layers, CoRR abs/1604.00825 (2016). arXiv:1604.00825.
URL http://arxiv.org/abs/1604.00825
[22] R. Barco, L. Nielsen, R. Guerrero, G. Hylander, S. Patel, Automated troubleshooting of a mobile communication network using bayesian networks, in: MWCN, 2002.
[23] R. M. Khanafer, B. Solana, J. Triola, R. Barco, L. Moltsen, Z. Altman, P. Lazaro, Automated diagnosis for UMTS networks using Bayesian network approach, IEEE Transactions on Vehicular Technology 57 (4) (2008) 2451–2461. doi:10.1109/TVT.2007.912610.
[24] S. Rezaei, H. Radmanesh, P. Alavizadeh, H. Nikoofar, F. Lahouti, Automatic fault detection and diagnosis in cellular networks using operations support systems data, in: IEEE/IFIP NOMS, 2016.
[25] E. J. Khatib, R. Barco, A. Gómez-Andrades, I. Serrano, Diagnosis based on genetic fuzzy algorithms for LTE self-healing, IEEE Trans. on Vehicular Technology 65 (3) (2016) 1639–1651. doi:10.1109/TVT.2015.2414296.
[26] G. Ciocarlie, U. Lindqvist, K. Nitz, S. Nováczki, H. Sanneck, On the feasibility of deploying cell anomaly detection in operational cellular networks, in: IEEE/IFIP NOMS, 2014.
[27] Q. Liao, S. Stanczak, Network state awareness and proactive anomaly detection in self-organizing networks, in: IEEE Globecom, 2015. doi:10.1109/GLOCOMW.2015.7414141.
[28] A. Terra, R. Inam, S. Baskaran, P. Batista, I. Burdick, E. Fersman, Explainability Methods for Identifying Root-Cause of SLA Violation Prediction in 5G Network, in: GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1–7. doi:10.1109/GLOBECOM42002.2020.9322496.
[29] S. Tang, J. Kong, B. Niu, Z. Zhu, Programmable Multilayer INT: An Enabler for AI-Assisted Network Automation, IEEE Communications Magazine 58 (1) (2020) 26–32. doi:10.1109/MCOM.001.1900365.
[30] Y. Wei, M. Peng, Y. Liu, Intent-based networks for 6G: Insights and challenges, Digital Communications and Networks 6 (3) (2020) 270–280. doi:https://doi.org/10.1016/j.dcan.2020.07.001.
URL https://www.sciencedirect.com/science/article/pii/S2352864820302418
[31] P. D. Grünwald, The Minimum Description Length Principle (Adaptive Computation and Machine Learning), The MIT Press, 2007.
[32] C. S. Wallace, D. L. Dowe, Minimum Message Length and Kolmogorov Complexity, The Computer Journal 42 (4) (1999) 270–283.
[33] Y. Yang, G. I. Webb, Discretization for naive-bayes learning: managing discretization bias and variance, Machine learning 74 (1) (2009) 39–74.
[34] S. García, J. Luengo, J. A. Sáez, V. López, F. Herrera, A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning, IEEE Transactions on Knowledge and Data Engineering 25 (4) (2013) 734–750.
[35] D. V. Carvalho, E. M. Pereira, J. S. Cardoso, Machine learning interpretability: A survey on methods and metrics, Electronics 8 (8) (2019) 832.
[36] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and regression trees., Kluwer Academic Publishers, New York (1984).
[37] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference and prediction, Springer, 2009.
[38] W.-Y. Loh, Fifty years of classification and regression trees, International Statistical Review 82 (3) (2014) 329–348.
[39] H. Peng, F. Long, C. Ding, Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy, IEEE Trans. Pattern Anal. Mach. Intell. 27 (8) (2005) 1226–1238.
[40] G. Brown, A. Pocock, M.-J. Zhao, M. Luján, Conditional likelihood maximisation: A unifying framework for information theoretic feature selection, Journal of Mach. Learn. Res. 13 (2012) 27–66.
[41] R. Cilibrasi, P. M. Vitányi, Clustering by compression, IEEE Trans. on Information theory 51 (4) (2005) 1523–1545.
[42] D. C. Hoaglin, F. Mosteller, J. W. T. (Editor), Understanding Robust and Exploratory Data Analysis, 1st Edition, Wiley-Interscience, 2000.
[43] D. Arthur, S. Vassilvitskii, K-means++: The advantages of careful seeding, in: ACM SODA, 2007.
[44] P. M. B. Vitanyi, F. J. Balbach, R. L. Cilibrasi, M. Li, Normalized information distance, arXiv preprint arXiv:0809.2553 (2008). arXiv:0809.2553.
[45] V. Mancuso, M. Peón Quirós, C. Midoglu, M. Moulay, V. Comite, A. Lutu, O. Alay, S. Alfredsson, M. Rajiullah, A. Brunström, M. Mellia, A. Safari Khatouni, T. Hirsch, Results from running an experiment as a service platform for mobile broadband networks in Europe, Computer Communications 133 (2019) 89–101.
[46] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, Z. Shi, The QUIC transport protocol: Design and internet-scale deployment, in: Proceedings ACM SIGCOMM, 2017.
[47] R. Marx, W. Lamotte, J. Reynders, K. Pittevils, P. Quax, Towards QUIC debuggability, in: Proceedings of ACM EPIQ, 2018.
[48] R. Marx, Main logging schema for qlog, Internet-Draft draft-marx-qlog-main-schema-02, Internet Engineering Task Force, Work in Progress (Nov. 2020).
URL https://datatracker.ietf.org/doc/html/draft-marx-qlog-main-schema-02
[49] H. Xiong, J. Wu, J. Chen, K-means clustering versus validation measures: A data-distribution perspective, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 39 (2009) 318 – 331. doi:10.1109/TSMCB.2008.2004559.
[50] S. Sieranoja, K-means properties on six clustering benchmark datasets, Applied Intelligence 48 (12 2018). doi:10.1007/s10489-018-1238-7.

[51] C. M. Bishop, Mixture density networks, 1994.
[52] G. Schwarz, Estimating the dimension of a model, The Annals of Statistics 6 (2) (1978) 461–464.
URL http://www.jstor.org/stable/2958889
[53] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), 1st Edition, Springer, 2007.
URL http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738