TTrees: Automated Classification of Causes of Network Anomalies with Little Data

Mohamed Moulay[†], Rafael Garcia Leiva[*], Vincenzo Mancuso[*], Pablo J. Rojo Maroni[‡], Antonio Fernandez Anta[*]
[†]University Carlos III of Madrid, Spain [*]IMDEA Networks Institute, Madrid, Spain [‡]Nokia Networks, Madrid, Spain
{mohamed.moulay, rafael.garcia, vincenzo.mancuso, antonio.fernandez}@imdea.org, pablo.rojo@nokia.com

***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. We present here TTrees (from Troubleshooting Trees), a practical and interpretable ML software tool that implements a methodology we have designed to automate the identification of the causes of performance anomalies in a cellular network. This methodology is unsupervised and combines multiple ML algorithms (e.g., decision trees and clustering). TTrees requires small volumes of data and is quick at training.**

Our experiments using real data from operational commercial mobile networks show that TTrees 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, hence enabling precise troubleshooting actions.

I. 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 [1]. Most importantly, new systems will require intelligent and automatic troubleshooting tools, which is the focus of our work. We resort to simple and unsupervised ML tools to build an interpretable and robust machine learning (ML) methodology, which we have implemented and tested with real data. Beside interpretability, a key advantage of our proposal is that it only requires access to little volumes of training data.

To provide customers with Quality of Service (QoS), many key performance indicators (KPIs) are continuously harvested to monitor network health. These indicators may measure the performance of radio, TCP, routing, etc. These measurements can be used to trigger troubleshooting actions. 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.

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 [2]. Multiple proposals exist for making fault detection and self-healing systems practical in mobile networks [3]. However, current self-healing troubleshooting proposals lack flexibility and do not scale well. There are newly-defined approaches based on ML, which use deep learning and neural networks (as black boxes) [4]. Unfortunately, these approaches lack interpretability, so it is not possible to understand the cause of a detected network 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 [5]. However, XAI does not help interpret AI decisions and conclusions *per se*. 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 [6].

In summary, troubleshooting remains a substantially manual procedure, in which highly skilled experts analyze alarms and statistics of performance indicators regularly to detect and diagnose the cause of problems in the network. In contrast, unskilled engineers can not even detect problems effectively [7].

A. Original Contribution of the Work

We propose an unsupervised methodology for the detection and clasification of network performance anomalies, and its software implementation, Troubleshooting Trees (TTrees). We join the ML research stream, while focusing on the automated detection and classification of possible network performance anomalies even with little volumes of measurements. Differently from existing ML proposals, we detect and classify anomalies by simply identifying the characteristics of what ML algorithms *cannot* learn/explain. For instance, a network that shows low throughput is not anomalous if this low KPI 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 (like decision trees), 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. In

fact, we prefer having lower accuracy, 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. Understanding and classifying these anomalous scenarios allow alerting the appropriate department to take corrective actions. Specifically, we use decision trees since they are transparent to inspection, hence interpretable [8].

We provide a Python open-source implementation of our methodology based on the Scikit-learn library [9] and a new library that we have implemented.[1]

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., metrics), but are also quite limited in terms of number of data samples. Thus, they are not suitable for commonly adopted deep neural networks and ML methods requiring complex training. The other dataset has been obtained in measurement campaigns with MONROE, an open-access platform for multi-homed experimentation with commercial mobile broadband networks across Europe [10]. With MONROE, we have access to a large dataset, although with a reduced number of metrics.

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 performance indicators, and that their unsupervised classification into network aspects was consistent. Additionally, they guided the development of a supervised implementation of the methodology, STrees, in which the KPIs were manually aggregated based on network aspects. The comparison of the outcomes of unsupervised and supervised methods with the same data has also been used to validate the methodology.

B. Organization of the Paper

The rest of the paper is organized as follows. Section II discusses the related work. Section III overviews the unsupervised methodology proposed. Section IV explains in detail how this methodology has been impemented as the TTrees system. Section V presents and analyzes the results TTrees achieves when it is applied to real datasets. We summarize and conclude the paper in Section VI.

II. Related Work

Early works on fault detection suggested the use of time series *regression* methods and *Bayesian networks*. For instance, Barco *et al.* [11] 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.* [12] 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 [13], 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 [14] 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 [12]–[14]. 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.* [15] 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 [16], 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.

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.

III. Overview of the Proposed Methodology

In this section, we briefly describe the framework, and the steps into which the automated unsupervised methodological process we propose is divided.

A. The Core Idea

Our methodology has been developed to find and interpret anomalies in a network, 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* and *minimum message length* principles [17], [18], according to which, learning from data is equivalent to compress data (i.e., describe data in the shortest possible way). Since classifying data points is a form of compression, we therefore conjecture that what a classifier cannot capture (hence the *misclassification* cases) are nothing but non-learnable behaviors, i.e., *anomalies*. For this reason, in our methodology, we apply ML classifiers to find anomalies by finding scenarios that cannot be learnt. Furthermore, we design the classifiers so as to be accurate and interpretable.

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

B. 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 the k performance indicators $(x_{i1}, \ldots, x_{ik})$ measured in scenario $i \in [1, n]$. In addition to these indicators, we assume that 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 characterized by the indicators $(x_{i1}, \ldots, x_{ik})$ and $(y_{i1}, \ldots, y_{i\ell})$.

C. Discretization

The first step of the methodology is to group the 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.[2] 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 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 among classes (although this may not be achievable in all cases).

D. Selection of Anomalous Scenarios

Once the data samples have been assigned to the classes based on their corresponding value of the target KPIs, we use them to train a classifier C_1. This classifier will use the indicators $(x_{i1}, \ldots, x_{ik})$ 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 $A \subset [1, n]$ of them. The scenarios in set A are the ones that we consider interesting, or *anomalous*: scenarios whose target KPIs can not be correctly classified by a properly trained classifier. 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 indicators 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., cell edge). This scenario should be correctly classified by classifier C_1 as bad, and a visual inspection of the radio indicators 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 performance indicators $\mathbf{x}_i$ are all good.

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

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

E. Selection of the Most Relevant Performance Indicators

Let $\mathbf{x}^A$ and $\mathbf{y}^A$ be the set of anomalous scenarios $\mathbf{x}_i$ 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 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 performance indicators for further analysis. Hence, in the next step of the methodology, we select a small subset $R \subset [1, k]$ of KPIs. The set R should contain only non-redundant indicators, 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 performance indicators of $\mathbf{x}^A$ by the amount of information they provide about $\mathbf{y}^A$. Then, using this ranking, select the top indicators in order, removing those that are redundant with the previously selected. This process stops when a certain number of indicators have been selected, or when the next indicator in the rank gives little information about $\mathbf{y}^A$.

F. Clustering Using the Most Relevant Indicators

The next step of the methodology is to use the indicators in R to classify the anomalous scenarios into meaningful types. We have observed empirically that the indicators in R belong to different network aspects (e.g., radio performance indicators, TCP performance indicators), 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 indicators is (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, *bad* and *good*. It is desirable to have balanced, compact and non-redundant clusterings. 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 vector of length α, with a value *bad* or *good* in each position. This 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. However, observe that the methodology cannot assign a meaning to a given value *bad* or *good* in the vector, since it is unsupervised. To do that, an expert should inspect the subsets and the clusters, and assign that meaning.

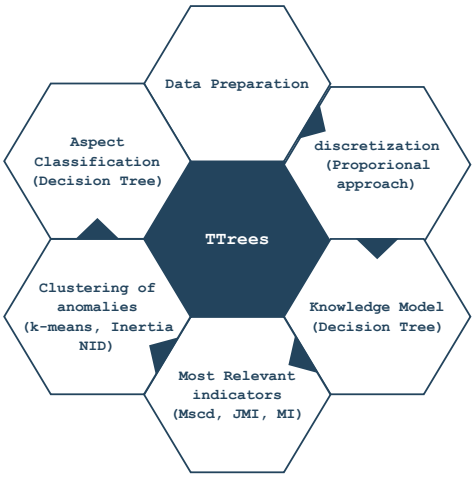


Fig. 1: Steps of TTrees (using multiple ML techniques): beginning with data preparation, proportional discretization, training of knowledge tree, selection of most relevant indicators, identification of anomaly clusters, and training of a network aspect anomaly classifier.

G. 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 indicators from R (although it is very likely that they will be used). It is desirable that this classifier is interpretable, so that an expert is able to understand why a scenario is considered to have issues in one particular network aspect.

If an expert has been involved in the process, it should have assigned a specific network aspect to each subset r_a, and a meaning to every value (*bad* or *good*) of every bit in the vector (the class output of the classifier). This information can be used to determine the issues each particular anomalous scenario has, and send it to the appropriate department.

H. 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.

In particular, consider a new scenario $(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$. Using classifier C_1 we determine that this is an anomalous scenario if the class C_1 assigned to $(x_1, \ldots, x_k)$ is not the one corresponding to the discretization classes of $(y_1, \ldots, y_\ell)$. 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 a binary vector of whether this scenario has issues in different network aspects. As mentioned before, if an expert identified the network aspects that correspond to each bit of the binary vector and which value of the bit identifies a type of issue, the vector can be used to report the anomalous scenario to the corresponding departments.

IV. TTrees

Here we present the implementation details of the tool TTrees that automatizes 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 embeds a number of standard tools for data analysis and ML, from discretization algorithms to clustering and classification.

A. 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 KPIs—the k indicators in $\mathbf{x}$ and the target KPI $\mathbf{y}$—that need to be observed (i.e., note that the given dataset might include more indicators 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 indicators are not reported. It also discards indicators that are constant through the dataset. Indicators 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.).

B. Discretization in TTrees

The selected target variable $\mathbf{y}$ is usually a numerical KPI, and its 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 variable 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 its class. Since TTrees is unsupervised, discretization uses progressive numbers as labels (e.g., "0" to "5").

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 [19], 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* [20]. 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.

C. 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 [21] 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 indicators $\mathbf{x}_i$. 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 may discard interesting samples.

The knowledge tree is build by applying the widely adopted Classification And Regression Tree (CART) algorithm [22], with the Gini impurity metric [23] 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 $\lfloor(\log_2 n)/2\rfloor$ since a deeper tree would suffer a high risk of overfitting [24]. We also limit the number of samples at the tree leaves to a minimum of five, to further reduce the risk of overfitting [23].

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 [23], 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 lest 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 not, an anomaly is identified.

D. Most Relevant TTrees Indicators

The next step consists in identifying a subset of indicators 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 indicators by computing how much information on the target $\mathbf{y}$ is contained in the indicator. For this task we have used the well known mutual information (MI) [25] and joint mutual information (JMI) [26] metrics 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) [27] which we call *miscoding* (Mscd). The miscoding of a sequence of $j \geq 1$ different indicators $\mathbf{x}^J = \langle \mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \dots \mathbf{x}_{i_j} \rangle$ with respect to the target $\mathbf{y}$ is defined as

$$\text{Mscd}(\mathbf{x}^J, \mathbf{y}) = \frac{1 - \text{NCD}(\mathbf{x}^J, \mathbf{y})}{\sum_{i_1, i_2, \dots i_j} (1 - \text{NCD}(\mathbf{x}^J, \mathbf{y}))}. \quad (1)$$

We propose Mscd because it measures the difficulty of reconstructing the target $\mathbf{y}$ from the indicators $\mathbf{x}^J$ and vice versa, which accounts for the redundancy in $\mathbf{x}^J$. In TTrees we start by computing a conditional redundancy matrix with the normalized compression distance of all possible pairs of indicators with respect to the target variable $\text{NCD}(\langle \mathbf{x}_i, \mathbf{x}_j \rangle, \mathbf{y})$, 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 indicators 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-indicator basis and is able to quantify the relevance of an indicator, but it fails to identify redundancy. JMI is instead used on indicator pairs, so that it allows to evaluate not only the relevance of an indicator but also the redundancy with another indicator. 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 indicator.

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 indicators 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 indicators much larger than the number of aspects to study (in the current implementation, α^2, where α is the number of aspects, which is fixed a priori).

E. Clustering of Anomalies in TTrees

Each pair of relevant indicators 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 [23], using the normalized values of the indicators obtained via RobustScaler [28]. 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 indicators, 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 plot. 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 indicators. In general, the cluster with issues is assigned a bit 1 in this aspect, and the other cluster is asigned a bit 0.

TABLE I: Brief description of the experimental datasets used in this work

ID	Dataset	# Samples	# Indicators	Families of performance indicators
#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

The problem is that we have potentially a large number of pairs of indicators (network aspects), and some of them might bring redundant information. Therefore, TTrees selects only a few pairs of indicators, so as to identify non-redundant descriptions of anomalies. In our current implementation we select $\alpha = 4$ indicator pairs, i.e., the 4 most relevant aspects to evaluate to understand the anomaly. Thus, the number of indicators passed from the previous step is $p = \alpha^2 = 16$. We do not use all possible indicators for a matter of practicality. For example, with $p = 120$ indicators, 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 indicator pairs are selected based on multiple metrics. First of all, TTrees computes the inertia score [29] of the clusters resulting from each indicator 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) [30]. Here the order matters, since if two indicator pairs yield redundant clusters, the indicator 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 an indicator 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).

F. Aspect Classification in TTrees

The α top indicator 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 a 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 bits with value 1 in a category 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 indicators, and not only based on the most relevant indicators 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.

G. 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.

V. Numerical 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 (i.e., the throughput) and restrict the analysis of anomalies to the case in which C_2 returns a better class than C_1, i.e., when the knowledge model predicts better results than discretized observed KPI.

A. Supervised ML-based Troubleshooting

The supervised version of TTrees, denoted as STrees, uses a supervised selection of the indicators that are most relevant for the detected anomalies. In particular, with the supervision of a network expert, STrees selects the most relevant indicators based on KPI families identified by the expert. The identified families range from radio-related indicators (e.g., radio buffer, radio coverage, etc.) to TCP-related indicators (e.g., RTT, TCP Window, TCP flags, etc.). The same steps described in Section IV-E for TTrees apply to STrees. However, the expert in the field of cellular networks carefully labels each cluster as anomalous data points with radio, buffering, or TCP issues. 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.

B. Datasets

We first analyze two data collections of cellular networking experiments used by Nokia in 2019 for network performance assessment of various European 4G networks. These two datasets contain drive-test information on TCP traffic experiments in which constant-size (3 MB) files are downloaded, and live web-pages as fetched with a mobile user device. Then, we further evaluate TTrees using a larger mobile broadband connectivity dataset collected with MONROE [31].

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 (date, time, location, network element information, etc.) and indicators (radio,

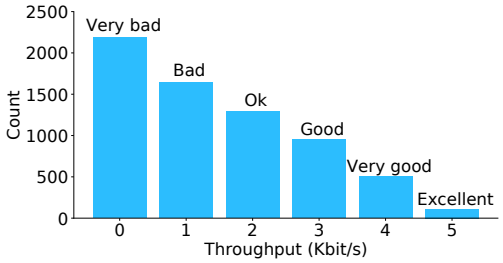


Fig. 2: Target KPI (throughput) distribution for Dataset #1 via the proportional discretization approach

TCP/IP, application, etc.) 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 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 performance indicators. Moreover, some of the MONROE dataset highlights include the ability to filter by test type, infrastructure, operator, vendor, and transmission technology.

In the next section, we use the first Nokia dataset to validate 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 second Nokia dataset is for HTTP webpage download performed live, over LTE networks, and one single operator as well. The MONROE dataset is for HTTP file download over LTE networks, and with one single operator, with tests anonymously conducted from labs and public buses. Table I summarizes the features of the datasets used.

C. 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. Fig. 2 illustrates the results obtained when applying the proportional discretization approach to this KPI in TTrees. Since there are 6690 samples, the number of categories used for this specific case is $\lfloor(\log_2 6690)/2\rfloor = 6$. 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 performance indicators 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 indicator (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

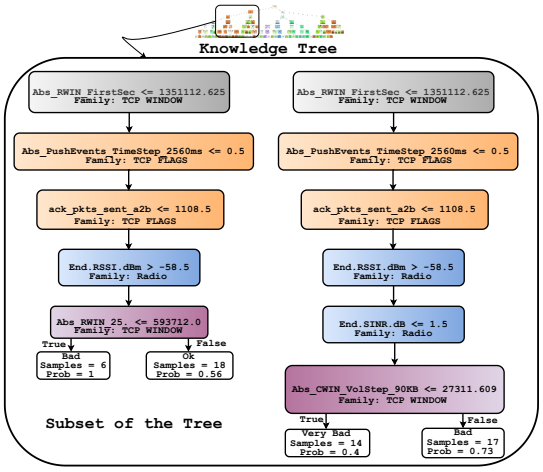


Fig. 3: Zoom into a subset of the knowledge tree built for Dataset #1. The figure also reports which family of indicators the branching variable belongs to (see Table I).

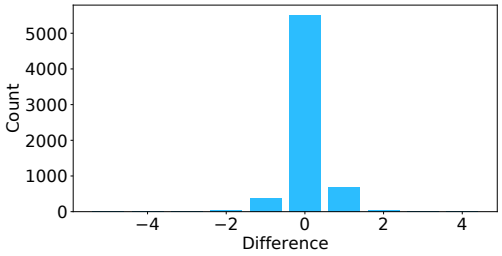


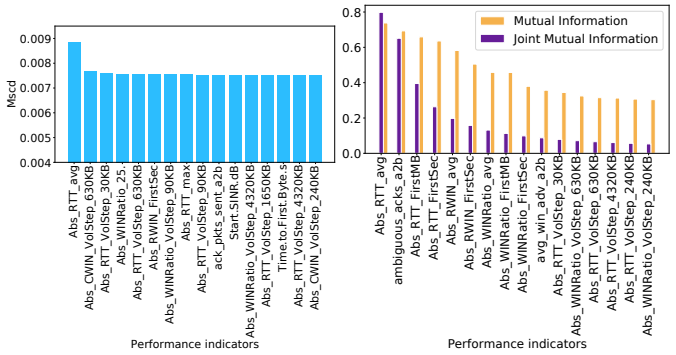
Fig. 4: Difference (predicted - discretized) between the class assigned by the knowledge tree and the discretized category for the target KPI of Dataset #1

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) 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 performance indicator, and whose family (radio, TCP window, TCP flags, etc.) is reported 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.[3]

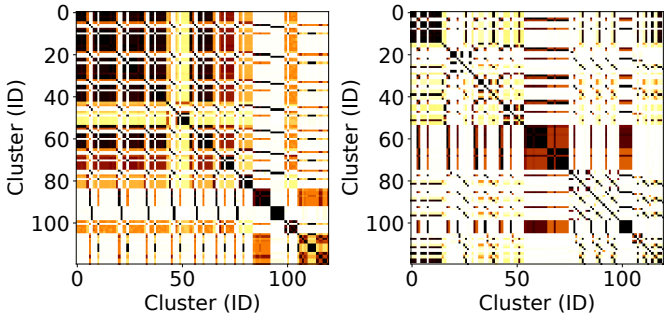
Fig. 4 shows that the knowledge tree classifies the target samples with high accuracy, although the number of cases in which the predicted class is better than the one observed in the discretization phase is not negligible ('-1' in the figure). TTrees selects these 740 anomalies to continue the analysis.

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



(a) Miscoding (b) MI and JMI

Fig. 5: Ranking of indicators according to their relevance to the description of anomalies for Dataset #1



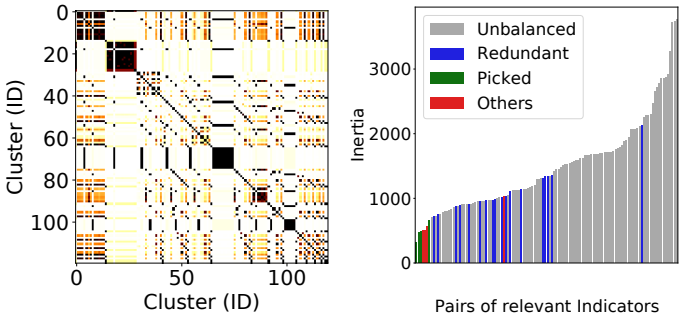
(a) MI-based selection (b) JMI-based selection

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

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 performance indicators to use is $\alpha^2 = 16$. Fig. 5 depicts the 16 most relevant performance indicators 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 performance indicators, while Mscd identifies a significantly different subset, which is due to the fact that Mscd accounts for redundancy of performance indicators while MI and JMI do not.

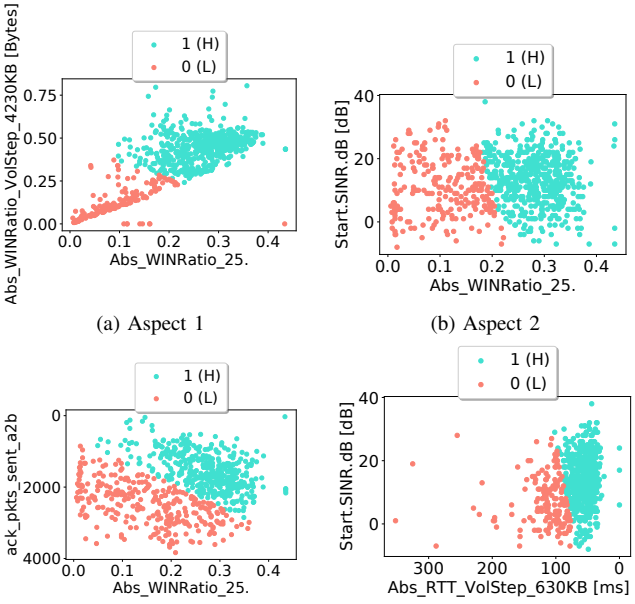
In Figs. 6 and 7(a) the redundancy of the binary clusters obtained with the 16 most relevant performance indicators is shown, for metrics MI, JMI, and Mscd, respectively. The heatmaps shown in the figures represent the NID metric between cluster pairs: the darker the color the lower the distance between 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 indicators whose combinations of clusters have lower redundancy. For this reason we will mainly consider the indicators selected with Mscd in the rest of this section.

Fig. 7(b) illustrates the inertia of the clusters obtained with the Mscd indicators, and the filtering process based on balance and low redundancy criteria. The output of this filtering process is a set of 4 network aspects (in practice, 4 pairs of performance indicators) used for binary clustering.



(a) NID of pairs of clusters built from indicators selected with Mscd (b) Inertia of clusters for all combinations of relevant indicators

Fig. 7: Network aspect selection based on the use of miscoding; lower inertia values of k-means clusters are preferred, unless clusters are redundant or unbalanced.



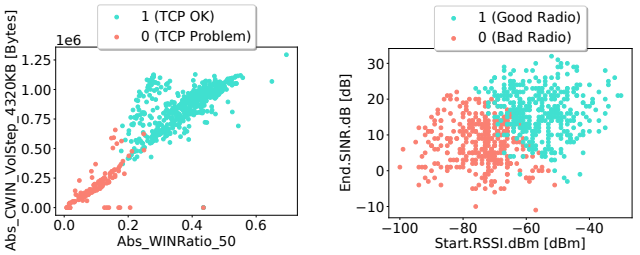
(a) Aspect 1 (b) Aspect 2

(c) Aspect 3 (d) Aspect 4

Fig. 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.

Due to lack of space, the visualization of the filtering results for the selection of relevant aspects obtained with MI and JMI is omitted. Results on the performance obtained with those metrics will be commented later and are reported in Table II.

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 groups. In this case, a Nokia cellular expert has identified for us the most relevant indicators, which led STrees to identify the network



(a) Aspect 1 (TCP Window)

(b) Aspect 2 (Radio coverage)

(c) Aspect 3 (RTT)

(d) Aspect 4 (Packet anomaly)

Fig. 9: Aspects obtained with STree for Dataset #1

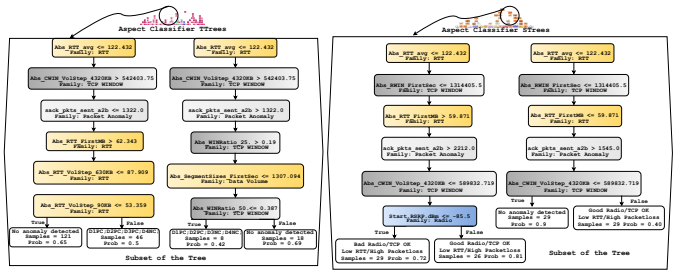


Fig. 10: A comparison between the supervised and unsupervised approach showing the resulting aspect classifiers

aspects visualized in Fig. 9. Aspects identified by TTrees and STrees are quite different, but in both cases they cover the same families (radio, TCP window, RTT, packet anomaly). The actual performance indicators 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 Abs_RTT_VolStep_630KB 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 Fig. 10, STrees is describing low RTT and high packet loss as well, deriving that result from the same performance indicator

TABLE II: TTrees performance with different datasets

	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

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 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.

D. Performance Evaluation

TTrees has been executed in the three different datasets. 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. Tests are done in classifiers C_1 and C_2 to get the metrics shown in Table II. The number of anomalous scenarios is 1152 and 6902, with 25 and 36 most relevant performance indicators, 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 indicators did not work well using MI and JMI for Dataset #2 (HTTP livepage DL), since all the pairs of indicators selected were redundant, and TTrees could not build C_2 (hence the entries "-" in Table II). 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 of multiple sizes. The fact that Mscd found meaningful network aspects is indeed impressive. For Dataset #1 and #3, all the three C_2 classifiers show similar classification performance (see Table II). However, the use of JMI and MI did not produce

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.

VI. Conclusions

In this paper, we have shown that it is possible to use interpretable ML to automatically identify the networking aspects that cause anomalous behaviors. Such anomalies cannot be immediately and directly explained by observing the network performance indicators. Their large number, even in the presence of a limited number of samples per indicator, makes it impossible to manually inspect and correlate. We have purposely developed a radically novel methodology and implemented it with Python, in TTrees. 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 in the network protocols the anomaly is rooted. In particular we have introduced a novel metric, miscoding, for the evaluation of redundancy and relevance of performance indicators. The final outcome of our proposed methodology is a set of easy-to-interpret classification rules, that, in case of performance anomaly, allow to automatically alert the appropriate departments for corrective actions. To do so, TTrees only needs little volumes of samples for the performance indicators. Indeed, TTrees allows to fully automatize network troubleshooting with high accuracy and fast training, as shown in this paper with the help of real data gathered from operational cellular networks in various countries.

Acknowledgments

Partially supported by Regional Government of Madrid (CM) grant EdgeData-CM(P2018/TCS4499, cofunded by FSE & FEDER) and Spanish Ministry of Science and Innovation grant ECID (PID2019-109805RB-I00, cofunded by FEDER). The work of V. Mancuso was supported by the Ramon y Cajal grant (ref: RYC-2014-16285) from the Spanish Ministry of Economy and Competitiveness.

References

[1] P. Yang, Y. Xiao, M. Xiao, and S. Li, "6G wireless communications: Vision and potential techniques," *IEEE Network*, vol. 33, no. 4, pp. 70–75, July 2019.

[2] 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, and M. J. Barros, "SELF-NET framework self-healing capabilities for 5G mobile networks," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 9, p. 1225–1232, Sep. 2016.

[3] A. Asghar, H. Farooq, and A. Imran, "Self-healing in emerging cellular networks: Review, challenges, and research directions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1682–1709, Apr. 2018.

[4] O. Loyola-González, "Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view," *IEEE Access*, vol. 7, pp. 154 096–154 113, Oct. 2019.

[5] A. Holzinger, M. Plass, K. Holzinger, G. Crisan, C. Pintea, and 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.

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

[7] T. Ogata, A. Takeuchi, S. Fukuda, T. Yamada, T. Ochi, K. Inoue, and J. Ota, "Characteristics of skilled and unskilled system engineers in troubleshooting for network systems," *IEEE Access*, vol. 8, pp. 80 779–80 791, Apr. 2020.

[8] N. Bostrom and E. Yudkowsky, "The ethics of artificial intelligence," *The Cambridge handbook of artificial intelligence*, vol. 1, pp. 316–334, Jul. 2014.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.

[10] 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, and M. A. Marsan, "Experience: An open platform for experimentation with commercial mobile broadband networks," in *ACM MobiCom*, 2017.

[11] R. Barco, L. Nielsen, R. Guerrero, G. Hylander, and S. Patel, "Automated troubleshooting of a mobile communication network using bayesian networks," in *MWCN*, 2002.

[12] R. M. Khanafer, B. Solana, J. Triola, R. Barco, L. Moltsen, Z. Altman, and P. Lazaro, "Automated diagnosis for UMTS networks using Bayesian network approach," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 4, pp. 2451–2461, July 2008.

[13] S. Rezaei, H. Radmanesh, P. Alavizadeh, H. Nikoofar, and F. Lahouti, "Automatic fault detection and diagnosis in cellular networks using operations support systems data," in *IEEE/IFIP NOMS*, 2016.

[14] E. J. Khatib, R. Barco, A. Gómez-Andrades, and I. Serrano, "Diagnosis based on genetic fuzzy algorithms for LTE self-healing," *IEEE Trans. on Vehicular Technology*, vol. 65, no. 3, pp. 1639–1651, March 2016.

[15] G. Ciocarlie, U. Lindqvist, K. Nitz, S. Nováczki, and H. Sanneck, "On the feasibility of deploying cell anomaly detection in operational cellular networks," in *IEEE/IFIP NOMS*, 2014.

[16] Q. Liao and S. Stanczak, "Network state awareness and proactive anomaly detection in self-organizing networks," in *IEEE Globecom*, 2015.

[17] P. D. Grünwald, *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[18] C. S. Wallace and D. L. Dowe, "Minimum Message Length and Kolmogorov Complexity," *The Computer Journal*, vol. 42, no. 4, pp. 270–283, Jan. 1999.

[19] Y. Yang and G. I. Webb, "Discretization for naive-bayes learning: managing discretization bias and variance," *Machine learning*, vol. 74, no. 1, pp. 39–74, Jan. 2009.

[20] S. García, J. Luengo, J. A. Sáez, V. López, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 734–750, Apr. 2013.

[21] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, no. 8, p. 832, Jul. 2019.

[22] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees." *Kluwer Academic Publishers, New York*, 1984.

[23] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*. Springer, 2009.

[24] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, no. 3, pp. 329–348, Dec. 2014.

[25] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, p. 1226–1238, Aug. 2005.

[26] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection," *Journal of Mach. Learn. Res.*, vol. 13, p. 27–66, Jan. 2012.

[27] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," *IEEE Trans. on Information theory*, vol. 51, no. 4, pp. 1523–1545, Apr. 2005.

[28] D. C. Hoaglin, F. Mosteller, and J. W. T. (Editor), *Understanding Robust and Exploratory Data Analysis*, 1st ed. Wiley-Interscience, 2000.

[29] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *ACM SODA*, 2007.

[30] P. M. B. Vitanyi, F. J. Balbach, R. L. Cilibrasi, and M. Li, "Normalized information distance," *arXiv preprint arXiv:0809.2553*, 2008.

[31] 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, and T. Hirsch, "Results from running an experiment as a service platform for mobile broadband networks in Europe," *Computer Communications*, vol. 133, pp. 89–101, Jan. 2019.