POLITECNICO DI TORINO
Repository ISTITUZIONALE

Supporting Telecommunication Alarm Management System with Trouble Ticket Prediction

Original
Supporting Telecommunication Alarm Management System with Trouble Ticket Prediction / Asres, M.W., Mengistu, M., Castrogiovanni, P., Bottaccioli, L., Macii, E., Patti, E., Acquaviva, A.. - In: IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. - ISSN 1551-3203. - 17:2(2021), pp. 1459-1469. [10.1109/TII.2020.2996942]

Availability:
This version is available at: 11583/2829652 since: 2020-11-22T14:04:09Z

Publisher:
IEEE

Published
DOI:10.1109/TII.2020.2996942

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

09 November 2026

Supporting Telecommunication Alarm Management System with Trouble Ticket Prediction

Mulugeta Weldezgina Asres, Million Abayneh Mengistu, Pino Castrogiovanni, Lorenzo Bottaccioli, Enrico Macii, Edoardo Patti and Andrea Acquaviva

***Abstract*—Fault alarm data emanated from heterogeneous telecommunication network services and infrastructures are exploding with network expansions. Managing and tracking the alarms with Trouble Tickets using manual or expert rule-based methods has become challenging due to increase in the complexity of Alarm Management Systems and demand for deployment of highly trained experts. As the size and complexity of networks hike immensely, identifying semantically identical alarms, generated from heterogeneous network elements from diverse vendors, with data-driven methodologies has become imperative to enhance efficiency. In this paper, a data-driven Trouble Ticket prediction models are proposed to leverage Alarm Management Systems. To improve performance, feature extraction, using a sliding time-window and feature engineering, from related history alarm streams is also introduced. The models were trained and validated with a data-set provided by the largest telecommunication provider in Italy. The experimental results showed the promising efficacy of the proposed approach in suppressing false positive alarms with Trouble Ticket prediction.**

***Index Terms*—Alarm Management, Trouble Ticket Prediction, Telecommunication, Data-Driven, Time Windowing, Feature Engineering, LightGBM, Machine Learning**

I. Introduction

Telecommunication networks contain thousands of interconnected components. These components are capable of generating error and status messages which leads to a large volume of network data. Hence, Telecommunications Service Providers use Information Technology (IT) systems, the Operations Support Systems (OSS), to communicate with network devices. OSS enables service providers to monitor, control, analyze, and manage the services on their network. When a network device detects a fault or an error within itself or related to its links to other devices or elements, it generates usually a large number of unsolicited alert messages and sends them to the alarm management system of the Network Operation Centers (NOCs). NOCs, central monitoring and control stations for telecommunication networks, primary deal with faults and performance management to maintain network efficiency and customer satisfaction. NOC operators are equipped with a complex alarm management platform (hereafter called *Alarm Management System (AMS)*) to visualize and manage alarms. The AMS presents relevant alarms to NOC operators after performing alarm filtration, correlation, categorization, and summarization on the unsolicited raised alarms (see Figure 1a). The NOC operators then analyze the presented alarms to perform the opening of Trouble Tickets (TT) so that the network technicians can fix the faults.

During the trouble ticketing, the operators may need to collect further information such as relationship or dependency from other alarms to ultimately discern the TT opening decision. However, such detailed troubleshooting relies, to a large extent, on the (most often unwritten) expert knowledge and experience of the NOC operators [1]. Therefore, as networks become more sophisticated, the telecommunication industry is facing an unprecedented heterogeneous amount of alarm data [2], [3]. The increase in variety, volume, and velocity of alarms from the expansion of networks have made it challenging to manually track faults and increases the demand for the deployment of highly trained operators [2]–[4]. Furthermore, the number of alarms, generated from the upcoming 5G networks, is expected to become overwhelming as the number of connected devices and their interconnections will increases significantly [4]. Consequently, telecommunication companies are investing in innovative intelligent technologies to enhance existing rule-based OSS [3], [5]–[8].

In industrial domains, knowledge extraction using data analytics and machine learning (ML) tools for fault alarm management have been proposed in literature [2], [5]–[7], [9]–[29]. Some of the proposed applications can be categorized into Alarm Correlation (ALCR) [2], [6], [11]–[19], Root-Cause Analysis (ALRC) [7], [17], [19], [20], TT Classification (TTC) [10], [14], [21]–[26] and TT Resolution Recommendation (TTRR) [23], [26]–[29] (see Section II-A for definitions). Generally, ALCR and ALRC tasks are carried out in the AMS before triaging the most urgent alarms to the NOC operator who issues appropriately labeled tickets (during TTC) to assist network technicians in fault resolution (during TTRR) [9]. ALCR, usually accompanied by ALRC, plays a crucial role in filtering the flood from False-positive alarms that do not need to be managed with TT while providing essential information for TTC and TTRR.

However, the integration of state-of-the-art approaches, for alarm flood handling in the telecommunication industry using ALCR [6], [11], [12], [14]–[16], [18], [19] algorithms, into existing AMSs is limited [2] due to issues related to inadequate accuracy, scalability and domain knowledge requirement [11], [12]. Consequently, most of the alarm filtering tools in the existing AMSs yet present thousands of less relevant alarms to the NOC operators [2], [3], [9], [11]. Though only a

M.W. Asres, M.A. Mengistu, L. Bottaccioli E. Macii and E. Patti are with Politecnico di Torino, Torino, Italy. Email: {name.surname}@polito.it
P. Castrogiovanni is with TIM, Torino Italy, Email: {name.surname}@telecomitalia.it
A. Acquaviva is with the University of Bologna Email: {name.surname}@unibo.it

small portion of presented alarms are managed with TT, the time and cost consumed by the operators to process and manage the presented alarms remain significant [2], [3]. The operators are encumbered by the flood of less relevant alarms hindering their ability to promptly manage urgent ones. This is due to the time wasted for filtration during the TT opening decision [11]–[15], [17], [19], [30] and it is exacerbated as the operator's trust on the AMS deteriorates due to the "cry wolf" effect [30], [31]. Moreover, alarm overload also increases the demand for deployment of highly skilled operators to meet fault maintenance time-constraints [2], [3].

In this study, we introduce an automated TT opening decision system that lies amid AMS and NOC operators, (illustrated in Figure 1b) to address the gaps in alarm flood control. We present Trouble Ticket Prediction (TTP) models used to predict whether a given presented alarm will be managed with a TT (True-positive alarm) or not (False-positive alarm). Our approach aims to significantly suppress the number of False-positive alarms by triage only the relevant alarms to NOC operators. We exploit supervised ML algorithms to learn from previous experiences on trouble ticketing besides alarm characteristics, unlike the unsupervised or Expert-Rule based ALCR methods. Our solution strives to classify semantically identical alarms, generated from various network elements of telecommunication mobile network (addressing 2G, 3G, and LTE network infrastructures and services alarms) from different vendors, for TT opening decisions using data analysis and ML algorithm. The aim is to automate and ameliorate the existing AMS (see Figure 1a), with scalable TTP system using ML (see Figure 1b).

The proposed system employs a sliding time-windowing and feature-engineering approach for alarm stream feature augmentation and extraction, and binary classifiers using gradient boosted decision tree algorithm (*LightGBM: Light Gradient Boosting Machine*) for modeling. The term "alarm stream" or "stream" refers to alarm data that are received over time. After applying data analysis techniques for data cleaning, correlation-based feature selection, and feature extraction, binary alarm classification models were trained for TTP. Performance evaluation vividly validates the capability of the trained TTP models in reducing alarm overload significantly while presenting the relevant alarms to the NOC operators. Finally, to demonstrate the significance of the proposed feature augmentation based TTP model, performance comparisons among several benchmark ML algorithms are carried out.

Handling presented alarms with automated TTP to triage the relevant alerts will assist the NOC operators in effectively managing a multitude of alarm streams. Therefore, the proposed system will have pertinence in enhancing the efficiency and competitiveness of service providers by alleviating the time and cost spent in the alarm management process. Finally, the key contributions of our work are highlighted below:

(i) Data-driven alarm classification for TT opening decision prediction, before TT triaging is carried out by NOC operators, to suppress False-positive alarm floods from AMS of heterogeneous telecommunication mobile networks.
(ii) Leveraging the TTP with feature augmentation and extraction methods using time-windowing and feature engineering from previously managed related alarm streams.
(iii) Characterization of the TTP with decision delay-time variations. Predicting the TT opening decision at the earlier-life time of the relevant alarms reduces the time-to-maintenance of the faults, thus reducing costs related to service interruptions [32], [33].

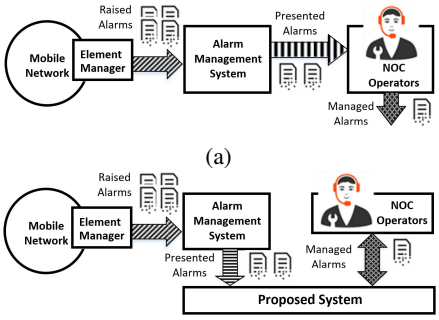


Fig. 1: Current (a) and proposed (b) architecture for alarm management system

The rest of this paper is organized as follows. Section II discusses background concepts and related works, Section III briefly highlights the data-set used in the study, Section IV presents the methodology and Section V discusses results of the experiments. Finally, Section VI offers the conclusion.

II. Background

This section discusses background concepts such as the working principles of AMS, the related works in fault alarm management, and the employed tools in our study.

A. Alarm Management System

The Alarm Management System (AMS) is responsible for storage, characterization, filtration, fault root-cause identification, and correlation analysis of the huge amount of alarms, messages, and events log collections from heterogeneous networks [12]. Figure 1a presents the high-level architecture diagram of a typical AMS. Events originating from heterogeneous networks, *Raised alarms*, are collected by their vendor-specific *Element Managers* and directed to the AMS for further processing, mostly using a rule-based reasoning paradigm [2]. The resulting processed alarms, *Presented alarms*, are displayed to NOC operators where True-positive alarms are managed by opening TT. AMS plays a vital role in handling low-level *Raised Alarm* floods using detailed domain knowledge from expert-defined rules, ontological, and topology databases [2].

Even though data-driven approaches such as statistical- and ML-based models were suggested in literature [6], [11], [12], [14]–[16], [18], [19], the existing AMS products still employ Expert Rule-based algorithms to accomplish most of their key functionalities [2]. Expert Rule-based algorithms provide higher accuracy through the incorporation of domain expertise in the decisions than the other techniques, yet they are limited in managing unseen conditions and large heterogeneous systems [11], [12]. Hence, most of the existing alarm flood filtration tools do not manage False-positives efficiently [2], [12]. Consequently, thousands of alarms per day

Solution	Purpose	ML Model	Features	TW	Target Alarms	Domain
[6], [12]	ALCR	**YES**	AOS	**YES**	Raised	**Telecom**
[18], [19]	ALCR	**YES**	AOS, **AA**	**YES**	Raised	**Telecom**
[13], [17]	ALCR	**YES**	AOS	**YES**	Raised	Manufacturing
[15]	ALCR	NO	AOS	**YES**	Raised	**Telecom**
[11]	ALCR	NO	AOS	**YES**	**Presented**	**Telecom**
[16]	ALCR	**YES**	**AA**	NO	**Presented**	**Telecom**
[14]	ALCR	**YES**	**AA**	NO	**Presented**	Enterprise IT
[28]	**TTP**	**YES**	TD	NO	Ticketed	Enterprise IT
[5]	**TTP**	**YES**	**AA**	NO	**Presented**	**Telecom**
Proposed	**TTP**	**YES**	**AA**	**YES**	**Presented**	**Telecom**

* ML - Machine Learning, TW - Time-windowing, AOS - Alarm Occurrence Sequence, AA - Alarm Attributes, TD - Ticket Description

TABLE I: Related works in False-positive alarm filtration

are yet burdening the NOC operators to perform decisions for discarding or silencing False-positive alarms before assigning proper TT.

Some of the relevant papers proposed to address the challenges of handling the flood of False-positive fault alarms with automated systems are summarized in Table I. However, considering the main tasks in automated alarm handling systems, the applications can be generally categorized as follows.

- *Alarm Correlation (ALCR)* is consolidating redundant correlated alarms. It is typically utilized to reduce related and spurious alerts (*Raised Alarms*), delivered by element manager nodes.
- *Alarm Root Cause Analysis (ALRC)* is determining of fault source or root cause of an alarm.
- *TT Prediction (TTP)* is classifying *Presented Alarm* to determine whether it will be managed by opening TT or not. Employing *TTP*, for false alarm suppressing, was also suggested in [5], [28] but in different scenarios. [5] proposed TTP to prevent repeated ticketing for IT customer premises alerts equipment. [28] recommended TTP for silencing False-positive alarms after ticketing based on their TT description. Yet, this may help fault resolving technicians but not the NOC operators because the main challenge of the alarm flood is before ticketing.
- *TT Classification (TTC)* is assigning appropriate TT class for True-positive alarms based on the required problem resolutions, priority or corresponding technical teams.
- *TT Resolution Recommendation (TTRR)* is suggesting troubleshooting solutions to ticketed alarms.

B. LightGBM: Light Gradient Boosting Machine

In this study, a machine learning algorithm, *LightGBM*, was employed to build the proposed TT prediction models. *LightGBM* is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithms. Although recent development of deep learning techniques achieved state-of-the-art performance in vision, speech and language domains, *GBDT (Gradient Boosting Decision Tree)* are currently popular production models in large scale applications [34] owing to their speed, interpretability, lower computational resource requirement and accuracy in training with mixed feature types [35]–[38]. In a *GBDT* ensemble model, decision trees are trained in sequence, *Boosting*, by fitting the negative gradients (also known as residual errors) in each iteration [36]. Existing state-of-the-art implementations of *GBDTs* include *XGBoost* [35], *LightGBM* [36], and *CatBoost* [37]. LightGBM uses leaf-wise learning to achieve a good balance between reducing the number of data instances and keeping the accuracy for learned decision trees. It was shown that *LightGBM* outperformed the other boosting tools in terms of efficiency (memory consumption and speed) and accuracy in various data-sets [36], [38].

C. Hyperparameter tuning

ML algorithms expose a set of parameters (hyperparameters) that must be configured before training. Tuning hyperparameters is a cumbersome yet crucial task, as the performance of the algorithms can be highly dependent on these parameters. Hence, several techniques have been proposed and applied to automate the tuning process. However, it has been shown that Bayesian optimisation based methods are faster and exceeds human experts at selecting hyperparameters on some competitive data-sets [39], [40]. Therefore, for hyperparameters tuning, we decided to employ an efficient *Bayesian Optimisation* based tool, i.e. *Hyperopt* [40].

III. Data-set Description

This section describes the data-set used in this study. The data was gathered from a data collection and correlation engine of AMS of TIM, the largest telecommunication provider in Italy. The data-set includes Presented alarms generated by 2G, 3G and LTE network equipment, focusing on the access segment of mobile networks. The data was collected for about six months during 2018 and 2019 and contains around 7.5 million *Presented* alarm stream events, recorded at every five minutes. However, only around 900 thousand unique alarms are composed. The remaining alarms events correspond to historized real-time instances of the unique alarms when there is an update in the alarm characteristics and/or Trouble Ticket status.

Each alarm record contains several attributes and the main ones are described in Table II. For the alarms managed with TT, additional information associated with how the alarm is processed by NOC operator, i.e., acknowledgment status, ticket status, time of closure, and closing remark, are contained in the TT attributes. Based on the alarm severity attribute, there are two main categories (each contributes around 48% of the *Presented Alarms*) worth mentioning, i.e. *Critical* and *Major*. *Critical* alarms, whose Trouble Tickets are expected to be issued in the first hours of alarms life, and *Major* ones, whose Trouble Tickets will typically be issued later. Alarm severity and first occurrence time are useful in prioritizing alarms to determine which alarm the operator should triage next [10]. Furthermore, in the data-set, the NOC operators managed only around 10% of the *Presented Alarms* with Trouble Ticket. The operators are immensely burdened to manually identify the relevant alarms from alarm overload presented by the AMS.

IV. Methodology

This section introduces the proposed ML-based TT prediction system. The basic modules of the proposed system

Attribute	Description
Feature-1	Type of the alarm
Feature-2	Correlation indicator filled by the AMS
Feature-3	Element Manager
Feature-4	Time in which the alarm is first observed by the AMS
Feature-5	Time in which the alarm is last observed by the AMS
Feature-6	Severity level of the alarm
Feature-7	Alarm root cause status
Feature-8	Site name
Feature-9	Site category
Feature-10	Description of apparatus function
Feature-11	Type of the network
Feature-12	Fault source
Feature-13	Possible probable causes which generated the alarm
Feature-14	Operation view flag
Feature-15	Number of attached children alarms
Feature-16	Number of repeated occurrence of the alarm
Feature-17	Minutes between the first and last occurrence of the alarm
Target	TT class label from TT identifier

TABLE II: Description of selected alarms attributes

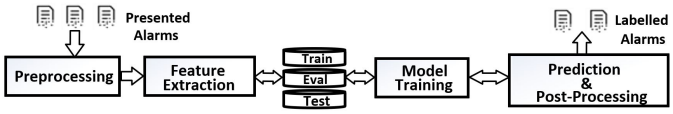


Fig. 2: The proposed AI-based TT prediction system

are portrayed in the architecture diagram in Figure 2. The approach embarks with data-set preparation implemented in the *Preprocessing* block. The inputs to this block are *Presented alarms* from the AMS. Then the processed alarms are fed to the *Feature extraction* block where the attributes are augmented to facilitate the model learning process. The resulting features are then split into *Training-* and *Test-set*. The *Training-set* is used by the *Training* block to fit and tune the ML models. The resulting models are then used by the *Prediction and Post-Processing* block for the TT prediction and performance evaluation on unseen alarms from *Test-set*. The following sub-sections describe in-depth each block in the proposed architecture.

A. Preprocessing

This section describes the data cleaning, feature selection and encoding operations performed on the alarms *Training-* and *Test-sets*.

1) Feature selection: the identification of most relevant features started with the consultation of domain experts which gave us the starting 43 attributes from which the number of alarm and TT attributes (which are not available to operators during real-time intervention on alarms) are 33 and 10 respectively. However, the resulting subset was further scrutinized with data analysis tools [41], [42] such as *Pearson Correlation* for the numerical features, and *Chi-Square Test* and *Association Mining* (*Cramer's V* and *Theil's U*) for the categorical features. The data analysis techniques facilitated the feature selection processes by quantifying the correlation/association and predictive capability of the attributes to the target variable and finally reducing the total relevant attributes to 18 (17 from alarm, and 1 from TT which is used for labeling) listed in Table II.

2) Categorical feature encoding: even though *Integer* and *One-hot encoding* are among the most popular categorical feature encoding techniques, they are opt-out due to semantic change, feature explosion, training speed and accuracy limitations. These shortcomings become more critical when the number and/or cardinality of categorical fields in the feature set increases, and the categories are non-ordinal. Particularly for high-cardinality categorical features, a tree built on one-hot features conduces to be unbalanced and needs to grow very deep to achieve good accuracy [36]. Employing advanced word embedding techniques such as *Word2Vec* [43] to encode the categorical features have not improved the classification performance in our experiments despite the computational cost. In this study, we used *LightGBM's* built-in categorical handling algorithm which sorts the categories according to the training objective at each split [36].

3) Tokenization: some of the alarm features comprise machine-generated description strings. To categorize such textual descriptions, text mining techniques are often employed to capture essential keywords while stifling irrelevant words. Tokenization is usually applied for text normalization to enhance the extraction of keywords while keeping the semantic structures [14], [21]. In our dataset, some of the textual alarm attributes contain information specific to a particular device, network, site, location, vendor, or date-time, such as IP address, serial number, node identification, vendor name, site name, date-time, and numbers. Thus, we have employed text token normalization to clean up such details by replacing them with *anonymous tokens* such as *ipaddress, serialno, nodeid, vendor, datetime, number, and location*.

4) Labeling: target class labels, for the TT prediction model training, were created using *Back-propagation* technique based on the TT fields of the alarm data-set described in Section III. Each alarm has multiple real-time historized events in which TT is assigned or revoked (in some cases) at some point in the real-time stream. Hence, the TT statuses of the final instance of the alarms are used for labeling. Then, the labels are assigned backward to all earlier instances of the alarms. This labeling method enables training the TT prediction models using attributes of the alarms in early life. The class labels of the target variable are given below.

- *Alarms with TT* (Positive class, referred also as true positive alarm): if TT is issued or opened for the alarm.
- *Alarms without TT* (Negative class, referred also as false positive alarm): if TT is not issued for the alarm.

B. Feature Extraction

In addition to developing a TTP model using predictor features from the attributes of the given alarms (see Section IV-A1), we also introduced feature set augmentation with engineered features from attributes of previously managed related history alarms given in Table III. The underlining concept is fault alarms originating from the same source within close time-window are most likely emanated from the same root causes [2], [6], [7], [11]–[20]. Hence, we propose a feature extraction method to take advantage of this concept and leverage the performance of the TTP models with augmented feature sets from related history alarms. Detailed illustration of the methodology for proposed TTP system is portrayed in Figure 3. Generally, the method composes two major steps:

time-windowing (*TW*) and feature-engineering (*FE*), hereafter referred together as *TWFE* (see Algorithm 1).

The time-windowing (see line 2 - 7 in Algorithm 1) incorporates features of related history alarm streams which arose from the same source as the target alarm within a given time-window. First, for target alarm A, history alarms, $hAlarms$, occurred within the range of TW are retrieved (see line 5). Then, N_h related alarms, $rAlarms$, which are generated from the same S, the origin of A, are filtered from the $hAlarms$ (see line 7). The extraction of features from the filtered alarms follows to form the engineered features, $rAlarmsFE$ (see line 8 - 9), which finally joins with the features of A. The size of the time-window ($TW = 15\,days$) and the number of previously managed related history alarms ($N_h = 15$) were learned heuristically by combining experts' domain knowledge, computation complexity, and model accuracy.

Although joining the $rAlarms$ using their temporal order can give quasi time-series data, the sequence of the history alarms may not have the same interpretation as classical time-series events [44], [45]. Thus, the *FE* is proposed to transform and aggregate the features from $rAlarms$ into a new compact feature set, $rAlarmsFE$. Besides, only the most relevant categorical features, sF, are selected from $rAlarms$ (see in Table III) using correlation-based feature selection (see Section IV-A1) and feature importance analysis (see Section IV-C3). When deciding the sF, further preference was also given to features with low cardinality due to the one-hot encoding feature expansion in the feature engineering (see line 7). The selected attributes of the history alarms, which are given in Table III, are included in sF except *TWFE_Feature-5* that corresponds to rD, relative time duration of occurrence of the $rAlarms$ to the target alarm A. Furthermore, using the previously managed related alarms allows us to include features from the TT attributes of $rAlarms$ (*TWFE_Feature-4 and -6*). Moreover, Figure 4 illustrates how the *TWFE* works and the major tasks also discussed below:

a) ExtractAlarmFeatures (line 4): after applying the feature selection techniques discussed in Section IV-A1, the predictor features from the target alarm A are prepared before applying *TWFE*. These features also hereafter called as $withoutTW$.

b) RetrieveRelatedAlarms (line 5): history alarms, $hAlarms$, which occurred within the time-window TW, before the occurrence time of alarm A, are retrieved from the previously managed alarm database.

c) AlarmSourceInfo (line 6): the source S of the fault alarm is formulated from the location attributes and network element identifications of the target alarm A.

d) FilterHistoryAlarms (line 7): related history alarms, $rAlarms$, which were generated from the same source as the target alarm A are filtered from the $hAlarms$.

e) FeatureEngineering (line 8): this block is the core section of the proposed feature-engineering approach. The *FE* extracts new transformed features from the selected features sF by analysing the occurrence of unique categorical values of the features in all the time windowed related alarms, $rAlarms$ (see line 11 - 21). The operation initialises with one-hot encoding like feature expansion for each feature F in sF,

Selected Features	Description
TWFE_Feature-1	Correlation indicator filled by the AMS
TWFE_Feature-2	Severity level of the alarm
TWFE_Feature-3	Operation view flag
TWFE_Feature-4	Alarm acknowledgment status
TWFE_Feature-5	Relative time of occurrence from the current alarm
TWFE_Feature-6	Trouble Ticket opening execution status

TABLE III: Selected features from related history alarms

which gives the occurrence scores, $occScores$, for each unique categorical values, $catValues$ (see line 16 in Algorithm 1, block 2 in Figure 4). Then, the $occScores$ are added after multiplied by a weighting function f_w to form the extracted features for each $catValues$ (see line 18 in Algorithm 1, block 3 and 4 in Figure 4). The weighting function is introduced to incorporate temporal importance using rD of the $rAlarms$, calculated in line 12. The extracted features from all the unique categorical values, $catValue$, are joined to prepare the engineered features for a given feature F, $rAlarmsFE_F$ (see line 19 in Algorithm 1). Finally, the extracted features from all features in the sF are joined to prepare the final engineered features, $rAlarmsFE$, from the $rAlarms$ (see line 20 in Algorithm 1). Moreover, the features of A, $withoutTW$, are augmented with the extracted features from $rAlarms$ using *FE*, $AlarmsFE$, to create the $withTWFE$ features (see line 9).

The feature score aggregation function (see line 18) is defined by f_{sc} and given in (1):

$$f_{sc}(v,\, occScores,\, rD) = \sum_{i=1}^{i=N_h} occScores(v_i, v) \times f_w(rD_i) \quad (1)$$

where $occScores(v_i, v)$ is occurrence score of a category value v at the i^{th} alarm in $rAlarms$ (2), and f_w is a weighting function (3). rD_i is the i^{th} relative duration i.e, the relative time of occurrence of the i^{th} alarm in $rAlarms$.

$$occScores(v_i,\, v) = \begin{cases} 1, \, if\, v_i = v \\ 0, \, if\, v_i \neq v \end{cases} \quad (2)$$

$$f_w(rD_i) = \frac{k}{(b + rD_i)} \quad (3)$$

where k and b are constant factors for normalization and biasing respectively. The proposed weighting equation is inversely related to the relative time of occurrence, rD, since most relevant history alarms occur closer to the target alarm in the time-line [6]. The constant parameters k and b control the sensitivity of the weighting and thus define the feature score distinctiveness. Their values were heuristically determined in our experiments. Using weighting temporal importance has also been suggested for time-series prediction in [44]. Nevertheless, unlike to our scenario, their weighting function is proposed for rewarding previous correct event predictions.

Finally, *TWFE* based feature augmentation increases the number of predictor features to 44 (17 from *withoutTW* and 27 from *rAlarmsFE*) as discussed in Section IV-A1 and IV-B.

C. Model Training

The Trouble Ticket prediction system, which predicts whether a given alarm is to be managed with TT or not,

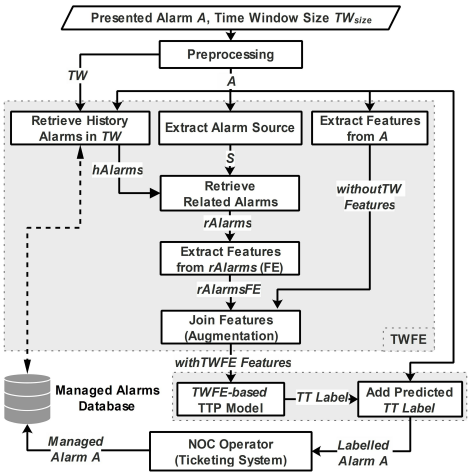
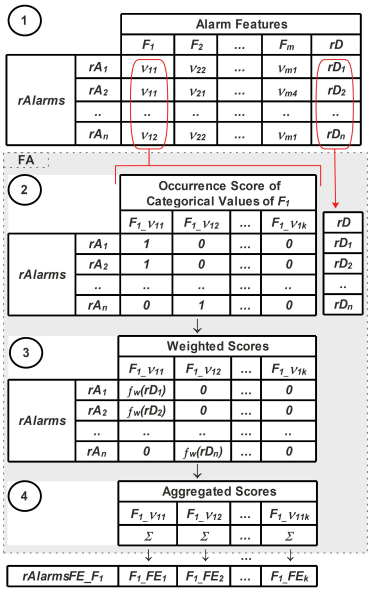


Fig. 3: Time-windowing feature-engineering flowchart



* rA_i - the i^{th} alarm from $rAlarms$, F_i - the i^{th} feature from sF, m - number of the selected features in sF, rD_i - relative duration of rA_i, $F_1_v_{ij}$ - the j^{th} categorical value of the i^{th} feature, k - number of unique categorical values of F_1, f_w - weighting function, $rAlarmsFE_F_1$ - engineered features from F_1 of all $rAlarms$, $F_1_FE_i$ - the i^{th} feature in $rAlarmsFE_F_1$,

Fig. 4: Example of feature extraction using feature-engineering

is implemented by training alarm classification ML models. Hence, the alarm classification models, which were trained to classify *Alarms with TT* and *Alarms without TT* under various scenarios, are used to demonstrate the efficacy of the proposed methods.

Furthermore, as explained in Section III, the data-set contains status of every alarm at every 5 minutes interval. Therefore, multiple models are trained and tested in order to demonstrate delay-time based TT prediction models which classify the alarms at delay-time of 0 to 60 minutes, with 5 minutes interval, after first arrival.

1) Training- and Test-set preparation: The prepared data-set is partitioned into Training- and Test-set using the time-series dimension, the *First occurrence* field. Due to sliding

Algorithm 1 Feature extraction using *TWFE*

procedure TIMEWINDOWFEATUREENGINEERING(A, TW_{size}, N_h, sF)
▷ A is the target presented alarm from *AMS*
▷ TW_{size} is size of time window TW
▷ N_h is maximum number of related history alarms
▷ sF is list of selected feature names
  $TW_{end} \leftarrow A_{OT}$ ▷ A_{OT} is a time of occurrence of A
  $TW_{start} \leftarrow TW_{end} - TW_{size}$
  $withoutTW \leftarrow$ EXTRACTALARMFEATURES(A) ▷ Features from A
  $hAlarms \leftarrow$ RETRIEVEHISTORYALARMS(A, TW_{start}, TW_{end})
  $S \leftarrow$ ALARMSOURCEINFO(A) ▷ Fault source location of A
  $rAlarms \leftarrow$ FILTERRELATEDALARMS($hAlarms, A, S, N_h$)
  $rAlarmsFE \leftarrow$ FEATUREENGINEERING($A, rAlarms, sF$)
  $withTWFE \leftarrow join(withoutTW, rAlarmsFE)$
  return A
procedure FEATUREENGINEERING($A, rAlarms, sF$)
  $rD \leftarrow$ RELATIVEDISTANCE($A, rAlarms$)
  $rAlarmsFE \leftarrow []$
  for $Feature\ F\ in\ sF$: **do**
    $rAlarmsFE_F \leftarrow []$
    $catValues, occScores \leftarrow$ FEATUREEXPANSION($F, rAlarms$)
    for $catValue\ v\ in\ catValues$: **do**
      $weightedScore \leftarrow$ AGGREGATESCORE($v, occScores, rD$)
      $rAlarmsFE_F \leftarrow join(F, v, weightedScore)$
    $rAlarmsFE \leftarrow join(rAlarmsFE_F)$
  return $rAlarmsFE$

Class Label	All (%)	Training (%)	Test (%)
Alarms with TT	9.30	8.60	11.14
Alarms without TT	90.70	91.40	88.86

TABLE IV: Target labels distribution

time-window construction of the features, time-series split without shuffling is adopted for the Training- and Test-set preparation. From the six months alarm data-set, the first four months was used for *training* (around 70%) the proposed models and remaining two months were used for *testing* (around 30%) evaluate the performance (see Table IV).

2) Hyperparameter tuning: hyperparameters tuning usually becomes sluggish when the number of parameters or candidate values is large, even with a guided optimized searching mechanism. Therefore, focusing on a small subset of parameters that have a significant influence on model performance is essential. Even though the implementation of *LightGBM* covers more than 100 hyperparameters[1], we selected a core set of parameters and found their optimal values, which achieved the best cross-validation score after training and validating the model on the Training-set. The criteria for choosing the core parameters was based on their potential for improvement in accuracy, speed and over-fitting control. Generally, the parameter tuning search consists of i) an estimator (the classifier), ii) a parameter space, a method for searching or sampling candidates, iii) a cross-validation scheme and iv) a score function. Some of the core parameters defined by *LightGBM* are *boosting*, *metric*, *num_leaves*, *min_data_in_leaf*, *max_depth*, *learning_rate*, *subsample_for_bin*, *lambda_l1*, *lambda_l2*, *feature_fraction*, *bagging_fraction* and *num_boost_round*

Moreover, there is a class imbalance in our Training-set which is dominated by negative class which drags the decision boundary of classifier toward the negative samples data space. However, using *AUC (Area Under the Curve Receiver Operating Characteristic Curve)*, as a cost function for the training, performed better in mitigating the class imbalance problem in our experiments. Finally, the same hyperparameters settings,

[1]https://lightgbm.readthedocs.io/en/latest/Parameters.html

tuned via the aforementioned approach using the *withoutTW* features, were adopted in all of our experiments.

3) Feature importance analysis: quantifying feature importance during ML model building has crucial contributions for model interpretability which facilitates the model adoption into the real systems. There are two feature importance measuring approaches in *LightGBM*: *split-based* (numbers of times the feature is used in a model) and *gain-based* (total gains of splits which use the feature). However, using these measures might be problematic as the measures are purely calculated on Training-set and it does not show the features relevance in the unseen data. Owing to the above issue, permutation based feature importance on Test-set has been suggested in literature [46], [47]. Another alternative method is *Permutation Feature Importance (PFI)* which is an intuitive, model-agnostic approach to estimate the feature importance of trained models by analysing the sensitivity of the model to random permutation (shuffled) of the feature values. In this study, the *PFI* is measured from the change in the models prediction accuracy when supplied with randomly permuted feature on the test-set.

D. Prediction and Post-Processing

This block starts by loading the model that has been trained in the *Training* phase to perform predictions on new test data. The classification results are stored and forwarded to the post-processing scripts for evaluation and generating meaningful representation of the results for display. The performance metrics and evaluation are detailed in Section V. At a given decision time-delay (up to one hour) and alarm severity, the predicted TT class labels along with classification confidence scores are presented.

V. Experimental Results and Discussion

In this section, performance evaluation of the conducted experiments with the proposed approaches, discussed in Section IV, is presented. The discussion compares binary classification of alarms for Trouble Ticket prediction performance reports on the *Presented Alarms* using the feature sets generated by *without using Time Windowing (withoutTW)* and *with Time-Windowing and Feature-Engineering (withTWFE)*. The reported performances are on the Test-set unless explicitly stated.

As discussed in Section IV, binary *LightGBM* classification models were trained using *GBDT* with AUC as a cost function and *Early-Stopping* for over-fitting handling. Performance metrics such as *Precision*, *Recall*, $F_1\ score$ *(F1)*, AUC, *Precision-Recall curve*, *Confusion Matrix* and *False Positive Rate* are employed to assess performance of the classifiers. We have also used *Average Precision* (AP) for performance comparison among different ML algorithms. AP summarizes the *Precision-Recall curve* as the weighted average of *Precisions* accomplished at each *Threshold*, with the change in *Recall* from the previous *Threshold* used as the weight, $AP = \sum_{i=1}^{n}(R_i - R_{i-1})P_i$ where R_i and P_i are *Recall* and *Precision* at i^{th} *Threshold*. Moreover, as pointed in Section III, the two relevant alarm severity categories, *Critical* and *Major*, have different sensitivity to the ticket decision timing. Thus, the discussion also incorporates the performance of the models on these severity scenarios.

Finally, performance comparisons of TTP models, trained by several ML tools [35]–[37], [48], and deep learning, are presented as benchmarks to evaluate the competence of the proposed *TWFE* and TTP model.

A. Alarm Classification for Trouble Ticket Prediction

Multiple classifiers were trained and compared, using the real-time alarm stream with different ticketing delay time. The performance scores of these delay-time based classifiers are discussed in Section V-A1. Furthermore, the classifiers trained at delay-time of 60 minutes has been chosen for a more detailed discussion on the performance gain of the proposed feature engineering technique (see Section V-A2).

1) Time-delay based classifiers: As explained in Section IV-C, to assess the impact of ticketing decision delay after the alarms first occurrence, thirteen models were trained to classify the alarms at delay-time of 0 to 60 minutes with 5 minutes interval. The performance of these delay-time based classifiers is illustrated in Figures 5 - 6. For each experiment, ten repeated trials have been carried out with random seeds. For each trial, the validation-set (10%) was kept the same during training *withoutTW* and *withTWFE*. The plots portray the mean scores along with 95% *Confidence Interval*.

Generally, the classification performance improves with the delay-time for both feature sets. The steepest boost in predicting the *Alarms with TT* occurred when the delay time is between 10 and 20 minutes. This corresponds to managing the *Critical Alarms* during the first half-hour of the alarm life. Our feature importance evaluation also indicated that the contribution of the duration feature (i.e. Feature-17) in the TT prediction becomes pertinent as the delay-time gets prolonged. The impressive accomplishment of the proposed TTP model is that it significantly filtered the False-positive alarms (93.6% correctly suppressed) and enhanced detection of True-positive alarms by 80% as compared to NOC operators at a one-hour time-delay. Furthermore, the proposed *withTWFE* technique greatly enhanced the classification performance as compared to the *withoutTW*.

The models trained *withoutTW* achieved, across the specified delay-times, mean score in the interval of [0.525, 0.599] with std (standard deviation) of [0.005, 0.009] in the $F_1\ score$, while the models trained *withTWFE* improve the performance to [0.600, 0.663] with std of [0.002, 0.004] (see Figure 5). The average performance improvement or gain (*absolute difference,* Δ) in the $F1$ ranges in the interval of [0.055, 0.075] with std of [0.005, 0.012] (see Figure 6). The *withTWFE* improved the mean AUC, with the gain of [0.021, 0.038], from [0.902, 0.926] to [0.930, 0.947] with std around 0.002. Another important result is the $F1$ of the most relevant alarms, i.e. *Critical Alarms*, reaches [0.647, 0.727] *withTWFE* having a gain of [0.051, 0.083] over *withoutTW*. The experimental results show proposed time-window history alarms-based feature extraction technique significantly enhanced the classification performance in predicting TT opening decision.

A detailed performance report using additional metrics at benchmark delay-times, i.e., at 0, 30, and 60 minutes, is

presented in Table V. The benchmark delay-times were chosen to take into account the delays applied by the AMS when presenting the *Critical* and *Major* Alarms. When the alarm classification is estimated instantly as the alarm arrives (at delay-time zero minutes), using *withTWFE* has accomplished a big improvement with a gain of 0.075 in $F1$ and 0.029 in AUC. This shows related history alarms provide useful information for the TT prediction.

2) Nominal classifiers: The above section showed the alarm features' predictive strength in alarm classification for TT prediction improves as the alarms stay longer in the system while the performance gain of the *withTWFE* over *withoutTW* stays consistent. One of these features is the alarm duration, because *Alarms without TT* generally tends to have a shorter duration than the *Alarms with TT*. Our analysis on the operators' waiting time before ticketing revealed that only around 40% of the ticketed alarms got their TT within an hour. Moreover, the most urgent critical alarms indicate faults that need to be resolved within hours[11]. Therefore, the models trained at delay-time of 60 minutes are chosen as the nominal classifiers for further detail performance comparison between the proposed models trained on single trial (see Figure 7 - 8). The performance comparison on the *Test-set* showed that the *withTWFE* outperforms *withoutTW* with gain of 0.08, 0.04 and 0.07 in the $Precision$, $Recall$ and $F1$, respectively, and the *False Positive Rate* reduced from 9.18% to 6.37%.

To evaluate the performance of the proposed TTP models on the trade-off between detecting True-Positive and suppressing False-Positive alarms, analysis on multiple *Thresholds* is required. Hence, the models were further evaluated with AUC and *Precision-Recall curve*. The model trained with *TWFE* achieved AUC of 0.945 with gain of 0.021. *Precision-Recall curve* is usually used to determine *a threshold T* which achieves the desired $Precision$ and $Recall$ given the attainable $F1$. The *Precision-Recall curve* accompanied by $F1$ contour lines and *F1-Recall curve* are given in Figure 7. The dotted annotations on the *Precision-Recall curve* are the T values, and the *F1-Recall Curve* portrayed $F1$ for the given $Recall$. The curves in Figure 7b show *withTWFE* enhanced the performance even at $T \neq 0.5$. *F1-Recall* curve and *F1* contour illustrate the T values, which give the max $F1$, are close to $T = 0.5$ with almost equal scores. This demonstrates the models are calibrated and also validates the comparison discussed in Section V-A1. The *Precision-Recall curve*, Figure 7, and the confusion matrices, in Figure 8, illustrate that the *withTWFE* gains its leverage by reducing the false positive predictions (increase by around 9% in *Precision*), as *Accuracy* on the negative class improves by 3%.

The mean *PFI* scores, after executing ten *PFI* trials per each feature on the *Test-set*, of the top-ten influential features, are depicted in Figure 9.Although Feature-13 (the probable causes) is the strongest feature followed by Feature-17 (the alarm duration), four features (prefixed with *TWFE*), extracted from the time-windowed related history alarms, have made it to the top-ten. This confirms the performance boost is from the extraction of new relevant predictor features from attributes of the history alarms despite the expansion of the feature set when using *TWFE*. Moreover, we found that the *PFI* ranking scores

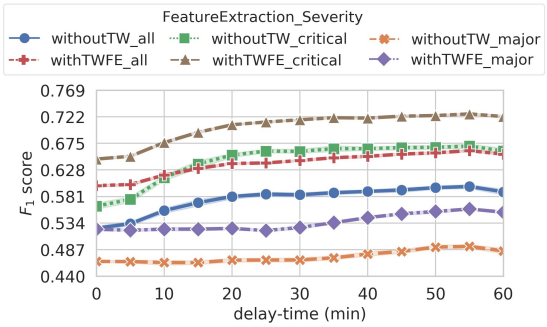


Fig. 5: $F_1 score$ of the time-delay based classifiers

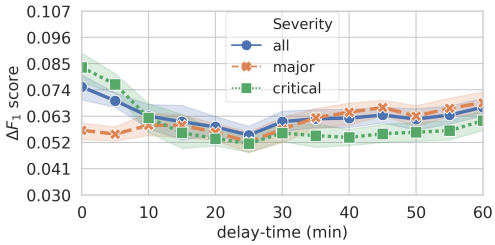


Fig. 6: $\Delta F_1\ score$ of the time-delay based classifiers

coincide with our feature analysis using the tools described in Section IV-A1. An asymmetric pair-wise association analysis with *Theilś U* and correlation analysis with *Pearson Correlation* for the nominal and continuous features, respectively, elucidate Feature-13 has the strongest association with the target variable. While the statistical correlation tools aid in identifying the relevant features before training a model, *PFI* is crucial in interpreting the trained model by distinguishing the prominent features during prediction. Therefore, combining feature analysis with *PFI* permits us to have a step-wise feature selection (forward-backward) mechanism, which facilitates the development of a better model.

B. Performance Comparison with Benchmark Models

To benchmark the proposed model performance, we employed other ML tools for TTP modeling. The tools are selected based on their excellent feasibility for the problem, i.e., particularly in handling class imbalance, many categorical features, high cardinality, and large data-set. However, only a few of the tools such as *LightGBM* [36] and *CatBoost* [37] offer a full support for categorical feature inputs. Therefore, we have devised a customized encoding technique, which transforms categorical features into numeric, to enable us to employ more ML tools such as *XGBoost* [35], *Random Forest* [48] and *Deep Neural Network (DNN)*. We employ One-hot and Word2Vec encoding schemes [43] for low cardinality (for features with less than 50 categorical values as suggested in [37]) and higher cardinality respectively. Moreover, for end-to-end modeling, we have also trained *DNN* with integrated embedded input layers[2,3] to handle categorical inputs directly without using external encoding schemes.

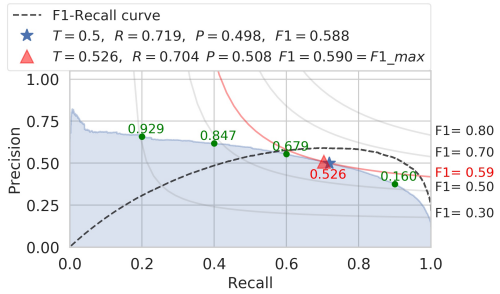
The benchmark models were trained with and without using the *TWFE* at one-hour delay-time, and Table VI summarizes their performance. Since the desired false or true positive rates

[2]https://keras.io/layers/embeddings/
[3]https://www.fast.ai/2018/04/29/categorical-embeddings/

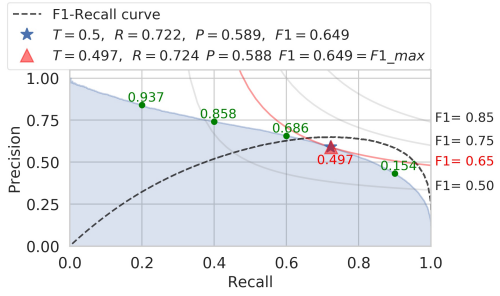
Feature Extraction	F1-score Mean ± Std	Precision Mean ± Std	Recall Mean ± Std	AUC Mean
delay-time (min) = 0				
withoutTW	0.525±0.007	0.451±0.007	0.628±0.019	0.902
withTWFE	0.600±0.004	0.562±0.008	0.644±0.009	0.930
Δ	0.075±0.009	0.111±0.011	0.016±0.024	0.029
delay-time (min) = 30				
withoutTW	0.584±0.006	0.506±0.009	0.691±0.013	0.921
withTWFE	0.645±0.004	0.601±0.011	0.695±0.010	0.942
Δ	0.061±0.007	0.095±0.013	0.004±0.012	0.021
delay-time (min) = 60				
withoutTW	0.589±0.008	0.501±0.013	0.716±0.016	0.923
withTWFE	0.656±0.004	0.601±0.009	0.722±0.006	0.944
Δ	0.067±0.007	0.100±0.013	0.006±0.017	0.022

*Δ - *Absolute difference between scores of withTWFE and withoutTW*

TABLE V: Classification performance on *Alarms with TT* at delay-time of 0, 30, and 60 minutes



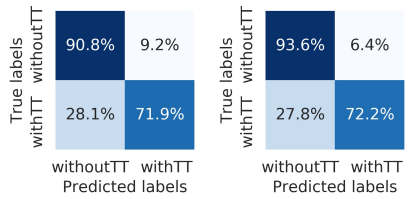
(a)



(b)

* *P- Precision, R- Recall, F1- F_1 score*

Fig. 7: *Precision-Recall curve* of the nominal classifier *withoutTW* (a) and *withTWFE* (b)



(a) (b)

* *withoutTT- Alarms without TT, withTT- Alarms with TT*

Fig. 8: Confusion Matrix of the nominal classifier *withoutTW* (a) and *withTWFE* (b)

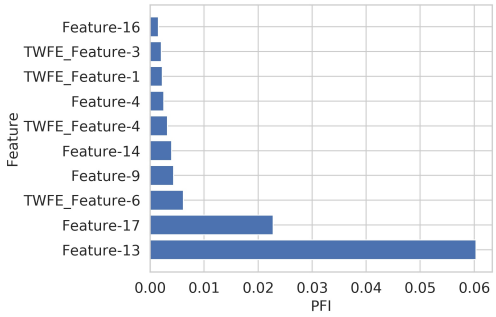


Fig. 9: *PFI* of nominal classifier *withTWFE*

Category Encoding	Model	Feature Extraction	*AUC*	*AP*	t_{trn}	t_{prd}	Δ*AUC*	Δ*AP*
Built-in Support	LGBM	*withoutTW*	0.93	0.60	**16.35**	2.18	0.018	0.107
		withTWFE	**0.95**	**0.71**	21.25	1.92		
	CB	*withoutTW*	0.92	0.57	117.42	0.70	0.021	0.109
		withTWFE	0.94	0.68	131.77	**0.69**		
One-hot and Word2Vec Encoding	LGBM	*withoutTW*	0.93	0.60	43.48	7.93	0.017	0.111
		withTWFE	**0.95**	**0.71**	53.41	7.93		
	CB	*withoutTW*	0.92	0.60	50.48	5.07	0.016	0.085
		withTWFE	0.94	0.69	55.98	5.08		
	XGB	*withoutTW*	0.91	0.54	195.89	7.05	0.025	0.144
		withTWFE	0.94	0.69	215.53	7.30		
	RF	*withoutTW*	0.92	0.59	98.58	5.78	0.017	0.110
		withTWFE	0.94	0.70	130.68	5.84		
	DNN	*withoutTW*	0.91	0.59	310.55	10.09	0.014	0.084
		withTWFE	0.93	0.67	579.30	9.78		
Embedded Input Layer	DNN	*withoutTW*	0.90	0.51	734.04	11.43	0.026	0.151
		withoutTW	0.93	0.66	744.19	10.76		

TABLE VI: Comparison with benchmark models

* *LGBM: LightGBM [36], CB: CatBoost [37], XGB: XGBoost [35], RF: Random Forest [48], DNN: (Architecture: 64-32-8 with Batch Normalization), t_{trn}: training time (sec), t_{prd}: prediction time (sec) including feature encoding time on test-set, Δ: gain of withTWFE over withoutTW, bold font: the best score*

are decided by the telecommunication operator requirement, we report *AUC* and *Average Precision* to provide a generalized performance of the classifiers across different decision thresholds. Generally, two relevant conclusions can be driven from the comparison. First, the performance of the models are very close except for the end-to-end *DNN* trained *withoutTW*. *LightGBM* and *CatBoost* are the best candidates due to their auto-handling of categorical features, speed, and accuracy. The other insight is that the *TWFE* feature augmentation has achieved consistent improvement (average $\Delta AUC = 0.02$ and $\Delta AP = 0.11$) in the TTP using the different ML algorithms.

VI. Conclusion

This study presents a Trouble Ticket Prediction system for telecommunication mobile network alarms using machine learning and feature augmentation. The proposed prediction system is intended to curtail the time and cost spent on alarm management which advances the efficiency and competitiveness of the service providers. The approach achieved encouraging outcomes in classifying the relevant alarms which need the opening of a TT. Therefore, the proposed models will support telecommunication operators to enhance their responsiveness in managing a large set of alarms. However, the models missed to detect around 28% of the True-Positive alarms at one-hour decision delay-time. Nevertheless, the missed alarms are still reduced by half as compared with the NOC experts at the same delay-time. Another limitation of the proposed supervised approach is that the TTP models may need retraining with an adequate number of examples to learn new alarms when deploying new network technologies with characteristics very diverse from those used during the training. Finally, the proposed feature extraction technique using *TWFE* achieved a perceivable improvement in the alarm

classification despite the negligible computation overhead. Hence, we believe the technique can also be adopted in similar Industry 4.0 data stream-based applications.

REFERENCES

[1] L. Hévizi, J. Fu, G. Magyar, and A. Rácz, "Creating a knowledge base for alarm management in a communications network," Jul. 9 2015, uS Patent App. 14/589,064.

[2] S. Salah, G. Maciá-Fernández, and J. E. DíAz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, 2013.

[3] C.-M. Chen, "Use cases and challenges in telecom big data analytics," *APSIPA Transactions on Signal and Information Processing*, vol. 5, 2016.

[4] N. H. Kumar and S. Baskaran, "Machine learning: The panacea for 5g complexities," *Journal of ICT Standardization*, vol. 7, no. 2, pp. 157–170, 2019.

[5] A. F. Fahmy, A. H. Yousef, and H. K. Mohamed, "The application of data mining for the trouble ticket prediction in telecom operators," in *2017 12th ICCES*. IEEE, 2017, pp. 227–232.

[6] O. Jukić and M. Kunštić, "Abcde-alarm basic correlations discovery environment," in *The 33rd International Convention MIPRO*. IEEE, 2010, pp. 528–533.

[7] M. Dror, Y. Lehmann, and G. Menahem, "Machine-learning and deep-learning techniques for predictive ticketing in information technology systems," May 2 2019, uS Patent App. 16/231,645.

[8] P. Tapia, "Artificial intelligence-based network advisor," 2017, uS Patent App. 15/615,688.

[9] Y. Temprado, F. J. Molinero, C. Garcia, and J. Gomez, "Knowledge discovery from trouble ticketing reports in a large telecommunication company," in *2008 CIMCA*. IEEE, 2008, pp. 37–42.

[10] B. Lee, A. Kapoor, R. Mahajan, B. S. Christian, and S. Amershi, "Classification of stream-based data using machine learning," Sep. 1 2015, uS Patent 9,122,995.

[11] J. Wang, C. He, Y. Liu, G. Tian, I. Peng, J. Xing, X. Ruan, H. Xie, and F. L. Wang, "Efficient alarm behavior analytics for telecom networks," *Information Sciences*, vol. 402, pp. 1–14, 2017.

[12] S. Sozuer, C. Etemoglu, and E. Zeydan, "A new approach for clustering alarm sequences in mobile operators," in *NOMS 2016-2016 IEEE/IFIP NOMS*. IEEE, 2016, pp. 1055–1060.

[13] V. Rodrigo, M. Chioua, T. Hagglund, and M. Hollender, "Causal analysis for alarm flood reduction," *IFAC-PapersOnLine*, vol. 49, no. 7, pp. 723–728, 2016.

[14] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *Proceedings of the 20th ACM SIGKDD*. ACM, 2014, pp. 1630–1639.

[15] Y. Cheng, I. Izadi, and T. Chen, "Pattern matching of alarm flood sequences by a modified smith–waterman algorithm," *ChERD*, vol. 91, no. 6, pp. 1085–1094, 2013.

[16] H.-C. Chao, C.-M. Hsiao, W.-S. Su, C.-C. Hsu, and C.-Y. Wu, "Modified adaptive resonance theory for alarm correlation based on distance hierarchy in mobile networks," in *2011 13th APNOMS*. IEEE, 2011, pp. 1–4.

[17] Y. Chen and J. Lee, "Autonomous mining for alarm correlation patterns based on time-shift similarity clustering in manufacturing system," in *2011 IEEE PHM*. IEEE, 2011, pp. 1–8.

[18] T. Li and X. Li, "Novel alarm correlation analysis system based on association rules mining in telecommunication networks," *Information Sciences*, vol. 180, no. 16, pp. 2960–2978, 2010.

[19] R. Costa, N. Cachulo, and P. Cortez, "An intelligent alarm management system for large-scale telecommunication companies," in *Portuguese Conference on Artificial Intelligence*. Springer, 2009, pp. 386–399.

[20] F. J. M. Velasco, "A bayesian network approach to diagnosing the root cause of failure from trouble tickets." *Artif. Intell. Research*, vol. 1, no. 2, pp. 75–85, 2012.

[21] J. Xu, H. Zhang, W. Zhou, R. He, and T. Li, "Signature based trouble ticket classification," *Future Generation Computer Systems*, vol. 78, pp. 41–58, 2018.

[22] C. Zeng, W. Zhou, T. Li, L. Shwartz, and G. Y. Grabarnik, "Knowledge guided hierarchical multi-label classification over ticket data," *IEEE TNSM*, vol. 14, no. 2, pp. 246–260, 2017.

[23] W. Zhou, W. Xue, R. Baral, Q. Wang, C. Zeng, T. Li, J. Xu, Z. Liu, L. Shwartz, and G. Ya Grabarnik, "Star: A system for ticket analysis and resolution," in *Proceedings of the 23rd ACM SIGKDD*. ACM, 2017, pp. 2181–2190.

[24] J. Xu, L. Tang, and T. Li, "System situation ticket identification using svms ensemble," *Expert Systems with Applications*, vol. 60, pp. 130–140, 2016.

[25] A. Maksai, J. Bogojeska, and D. Wiesmann, "Hierarchical incident ticket classification with minimal supervision," in *2014 IEEE ICDM*. IEEE, 2014, pp. 923–928.

[26] S. Amershi, B. Lee, A. Kapoor, R. Mahajan, and B. Christian, "Human-guided machine learning for fast and accurate network alarm triage," in *Twenty-Second IJCAI-11*, 2011.

[27] A. Watanabe, K. Ishibashi, T. Toyono, K. Watanabe, T. Kimura, Y. Matsuo, K. Shiomoto, and R. Kawahara, "Workflow extraction for service operation using multiple unstructured trouble tickets," *IEICE Transactions on Information and Systems*, vol. 101, no. 4, pp. 1030–1041, 2018.

[28] W. Zhou, L. Tang, C. Zeng, T. Li, L. Shwartz, and G. Y. Grabarnik, "Resolution recommendation for event tickets in service management," *IEEE TNSM*, vol. 13, no. 4, pp. 954–967, 2016.

[29] R. Potharaju, N. Jain, and C. Nita-Rotaru, "Juggling the jigsaw: Towards automated problem inference from network trouble tickets," in *Presented as part of the 10th USENIX NSDI)*, 2013, pp. 127–141.

[30] P. Goel, A. Datta, and M. S. Mannan, "Industrial alarm systems: Challenges and opportunities," *Journal of Loss Prevention in the Process Industries*, vol. 50, pp. 23–36, 2017.

[31] W. Xiong, J. Wang, and K. Chen, "Multivariate alarm systems for time-varying processes using bayesian filters with applications to electrical pumps," *IEEE TII*, vol. 14, no. 2, pp. 504–513, 2017.

[32] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine learning for predictive maintenance: A multiple classifier approach," *IEEE TII*, vol. 11, no. 3, pp. 812–820, 2014.

[33] W. Yu, T. S. Dillon, F. Mostafa, W. Rahayu, and Y. Liu, "A global manufacturing big data ecosystem for fault detection in predictive maintenance," *IEEE TII*, 2019.

[34] J. Zhu, Y. Shan, J. Mao, D. Yu, H. Rahmanian, and Y. Zhang, "Deep embedding forest: Forest-based serving with deep embedding features," in *Proceedings of the 23rd ACM SIGKDD*. ACM, 2017, pp. 1703–1711.

[35] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD*. ACM, 2016, pp. 785–794.

[36] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NIPS*, 2017, pp. 3146–3154.

[37] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *arXiv preprint arXiv:1810.11363*, 2018.

[38] X. Ma, J. Sha, D. Wang, Y. Yu, Q. Yang, and X. Niu, "Study on a prediction of p2p network loan default based on the machine learning lightgbm and xgboost algorithms according to different high dimensional data cleaning," *Electronic Commerce Research and Applications*, vol. 31, pp. 24–39, 2018.

[39] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[40] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in Science Conference*. Citeseer, 2013, pp. 13–20.

[41] A. Agresti, *An introduction to categorical data analysis*. Wiley, 2018.

[42] A. Taha and A. S. Hadi, "Pair-wise association measures for categorical and mixed data," *Information Sciences*, vol. 346, pp. 73–89, 2016.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[44] S. Zhang, S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Deep learning on symbolic representations for large-scale heterogeneous time-series event prediction," in *2017 IEEE ICASSP*. IEEE, 2017, pp. 5970–5974.

[45] Q. Zhu, Z. Chen, and C. S. Yeng, "A novel semi-supervised deep learning method for human activity recognition," *IEEE TII*, pp. 3821–3830, 2018.

[46] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[47] N. Huang, G. Lu, and D. Xu, "A permutation importance-based feature selection method for short-term electricity load forecasting using random forest," *Energies*, vol. 9, no. 10, p. 767, 2016.

[48] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.