Journal of Engineering Science and Technology
Vol. 16, No. 5 (2021) 4263 - 4282

THE AUTOMATED MACHINE LEARNING CLASSIFICATION APPROACH ON TELCO TROUBLE TICKET DATASET

FAUZY CHE YAYAH*, KHAIRIL IMRAN GHAUTH, CHOO-YEE TING

Faculty of Computing & Informatics, Multimedia University, Cyberjaya, Malaysia
*Corresponding Author: akunyer@gmail.com

Abstract

This paper presents automated machine learning for solving a practical problem of a telco trouble ticket system. In particular, the paper's focus is on the classification of early resolution code from the trouble ticket dataset. The number of trouble tickets is rising every year due to the new challenges from the digital world. It is a challenging job to evaluate the vast content of the trouble tickets manually. Past trouble ticket contains essential information about the root cause and the resolution to each problem. The main contribution of providing the early resolution code for each new trouble ticket can significantly reduce Mean Time to Restore (MTTR) for the telco operation, thus improves customer satisfaction and minimize telco business and operation costs. The research methods include the existing traditional model and its modification towards the best accuracy. Automated Predictive Engine (AutoPE) improves the current traditional engineered model's classification accuracy from 5% to 38% when using the optimal performing solution by implementing AutoML and Grid Search. This solution uses multiple classifiers such as Random Forest, Deep Learning, Gradient Boosting, XGBoost, and Extremely Randomized Trees classifiers on a set of features based on various telco serviceable broadband zones and sampling size. Finally, compared to the baseline existing traditional engineered model, the best performing solution also improves the quality of classification for the early resolution code for the telco trouble tickets dataset.

Keywords: Automated ML, Classification, Grid search, Trouble ticket.

1. Introduction

Numerous hardware components are managed in telecommunications networks: servers, repeaters, hubs, routers, modems, switches, cables, gateways, bridges, sensors, interface cards, etc. There are so many possible links between components, and even more so when they are heterogeneous. For example, the cooling system directly links all the hardware installed in the room that regulates room temperature; a past event's failure may affect the current hardware output or may result. A fibre cut that disconnects many dependents in the mobile base stations and affects several clients is another common cause. This problem prohibits traditional programming approaches from explicitly associating past accidents with the root cause [1]. The incident management systems usually referred to as trouble ticket systems [2], are practised as guidance by the technician to link all accidents. So, it is up to the technician to recognize these conditions and resolve them immediately. After many related accidents, technicians have invested a lot of time and money researching the real issue after correctly determining the root causes.

Meanwhile, these time inquiry attempts affect all other customers, making things more difficult. Machine learning approaches continue to be applied in a trouble ticket environment to assess new insights and automate the task in recent years. A major accident and thousands of incidents that can occur every day can impact vast internet infrastructure and spread to many customers. Such cases, however, cannot delay the steps taken, and necessary action must be taken. This paper extends the methodology of advanced machine learning, such as automated machine learning, to solve the industrial problems in the telco trouble tickets system. The methodology consists of five steps: data collection, evaluation of the dataset, feature engineering, machine learning modelling and classification, and the results, including the potential improvement. The methodology is applied in the area of trouble tickets system to cater to these problems.

This paper's contribution is the applied contribution of using the methodology of development to classify the resolution code for the telco trouble tickets using the automated machine learning framework. The investigated solution included the modification of the existing manual traditional machine learning algorithm application. The results from the analysis indicate a significant improvement in classifying the early resolution code. The accuracy of classification results improves from 5% to 38%, depending on the zone and sampling. The rest of the paper is organized as follows. Section 2 is a brief introduction to the related works. Section 3 is the methodology applied in this study. Results in detail are discussed in Section 4, and Section 5 concludes this study.

2. Related Works

2.1. Automated machine learning (AutoML) and traditional ML

In recent years, the automated machine learning [3] method has had a revival in the A.I. community, producing more powerful computational algorithms. Demand for experts in machine learning has outstripped availability amid the massive influx of people joining the industry. Developing user-friendly machine learning tools that non-experts can use to address this disparity has made considerable progress. The first efforts to streamline machine learning were to create open, coherent interfaces for various machine learning algorithms. All this makes AutoML an efficient A.I.

tool to classify early resolution code in this study. This work aims to deliver fully automated machine learning processes to improve trouble ticket classification [4] accuracy, reduce classification time, and enhance the model's quality. The AutoML decreases the steps into two-phase steps, as shown in Fig. 1. However, the traditional machine learning method needs different teams, beginning of data acquisition until predictions phase, affecting too much time before the proper model is successfully constructed.

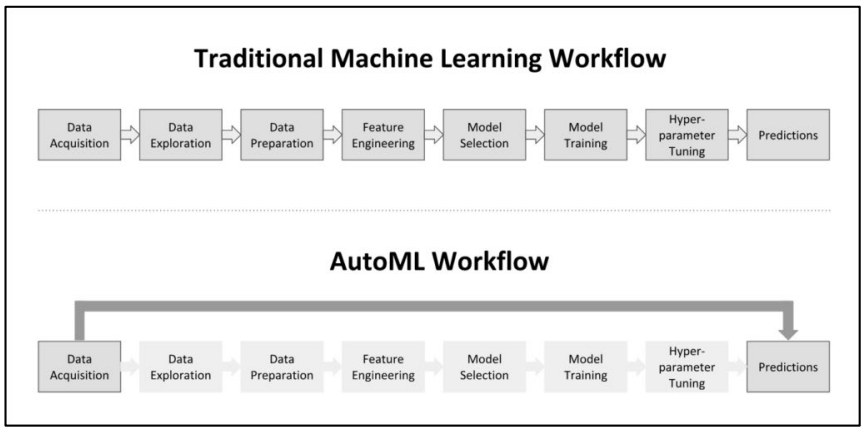

Fig. 1. AutoML and traditional ML.

Throughout this study, the AutoML approach is applied to overcome previously described problems with multiple classification algorithms within the telecommunications network. As this is a real challenge to be used in an existing trouble ticket system of several thousand daily occurrences, the solution approaches performance issues of significant attention because response times are essential.

2.2. Early resolution code motivation

The trouble ticket system [5] manages accidents or faults on network components or customer related. Thousands of incidents occur every day, involving several network components. All network components must work together, and any breakdown in this connection would cause network services failure. The regular flow of events inside trouble ticket systems is as follows; if the monitoring module detects an anomaly within the network, the alarm is automatic will be triggered. The trouble ticket system produces a new ticket that displays the network components' anomaly, as displayed in Fig. 2.

The ticket is sent directly to a team of trained engineers and technicians reviewing the ticket, and they attempt to fix it immediately. If the trouble tickets system can recognize the origin is the same symptom error code, creating a specific master-detail relationship may help locate the real cause by providing the early resolution code. This problem is critical to be addressed. Therefore, automated machine learning to the trouble ticket system is required in this study to minimize these errors. The motivation by having early resolution code classification helps the technicians focus on the area with higher chances to resolved it like before.

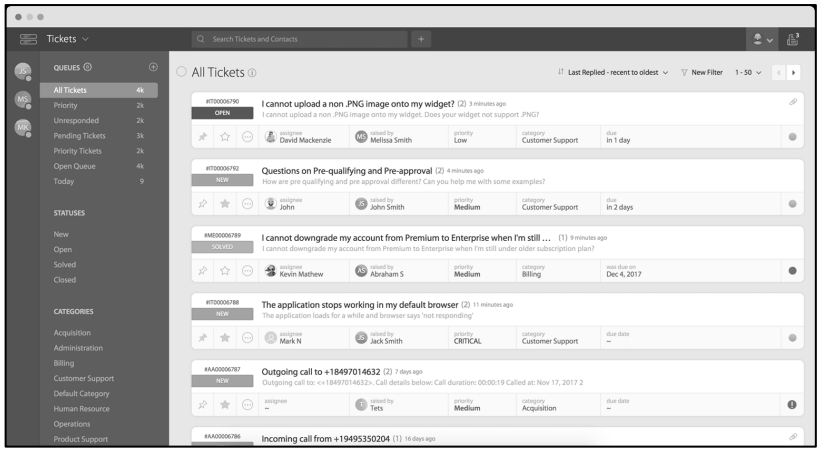

Fig. 2. Example of a trouble ticket system interface.

Some other previous works and gap analysis that applied machine learning in the trouble tickets study are summarized in the following Table 1.

Table 1. Traditional ML on trouble ticket research gap.

Journal	Algorithm	Limitation / Gap
Illuminating Trouble Tickets with Sublanguage Theory (2006) [6]	LibSVM	i. Implementing with a single traditional algorithm for such a massive task without parallelism. ii. SVM algorithm is not suitable for large data sets. SVM does not perform very well when the data set has more noise, or target classes are overlapping
Knowledge Discovery from Trouble Ticketing Reports in A Large Telecommunication Company (2008) [7]	Decision Tree, Bayesian Network	i. Implementing multiple traditional algorithms manually without any hyperparameter settings. ii. No parameter tuning was found.
Troubleminer: Mining Network Trouble Tickets (2009) [8]	Hierarchical clustering	i. Implementing a single traditional algorithm for clustering telco trouble tickets without achieving parallelism. ii. No option for parameter tuning was found. iii. Hierarchical clustering method cannot determine how many clusters is the most optimal.
A Bayesian Network Approach to Diagnosing the Root Cause of Failure from Trouble Tickets (2012) [1]	Bayesian Network	i. Implementing a single traditional algorithm for the classification of telco trouble tickets without achieving parallelism. ii. No significant explanation for the conditional probability links between different nodes makes the computational burden.
Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets (2015) [9]	Aho-Corasick	i. Implementing a single traditional algorithm for telco trouble tickets' clustering with Natural Language Processing (NLP).

		ii. Large keyword processing requires parallel processing, which does not exist in the study.
Predicting Network Faults Using Random Forest and C5.0 (2018) [10]	Random Forest, C.50	i. Comparison of two traditional algorithms for prediction of a telco network fault. ii. No option for parameter tuning was found. iii. Does not apply any parallelism technique due to the limitation of the traditional algorithm capability.

Most of the previous works related to the trouble tickets applied various traditional machine learning algorithms to solve the dataset's multiclass problem. Multiclass problems are not recommended to be solved by a single traditional algorithm alone, leading to misapplication for some circumstances. Some of them have not applied any feature selection based on the objective of the research. Therefore, the nature of the trouble tickets dataset requires multiple algorithms to solve it simultaneously. The problem with feature engineering is that it needs high domain knowledge because it includes designing new features. Implementing automated machine learning is essential that these problems can be avoided and improved by selecting the best algorithm compatible with the dataset and the features selected.

3. Methodology

3.1. Dataset

The current dataset used in this study was collected from four different sources, including Customer Internet Service Quality (CISQ) [11], Customer Internet Bandwidth (CIB) [12], Customer Trouble Ticket (CTT) [5], and Customer Profiles (CP) [13]. CTT collection contains various types of details relating to any network fault. The CISQ dataset contains several types of customer site support information. The CP dataset offers a detailed overview of the consumer's business based on their demographics, experiences, and product preferences. Finally, the CIB dataset provides different sources of knowledge on a particular internet connection's capacity and how data is transmitted from point to point. The summary of the data source for this research is illustrated in Fig. 3.

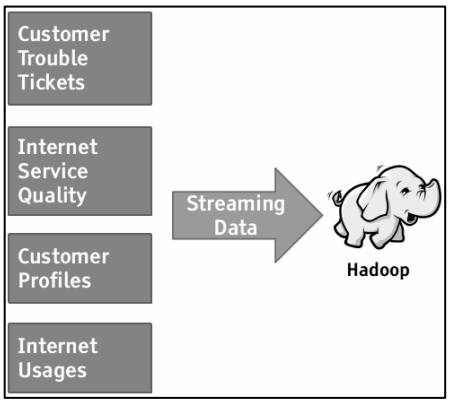

Fig. 3. Data consolidation method.

Additional information, such as the raw dataset, is streamed into the Hadoop Distributed File System (HDFS) repository [14] and updated daily for every hour. It is essential to address the sizing issues stored inside Hadoop due to Hadoop's ability to store unlimited data [15]. The consolidated dataset version is automatically created by Hadoop and stored in stratified sampling [16]. The predictive engine applies the stratified samples with the testing and validation process. The predictive values are applied for the next phase action, such as visualization and reporting. In Hadoop, each data aggregation function is performed to maintain the research dataset's enormous size, using its modular scalability architecture, faster computation, and embracing its tolerance to failure features. As Hadoop updates its latest dataset, the predictive engine will automatically apply it to classify the trouble ticket resolution codes. Due to a few indicators, such as training dataset consistency and sampling process, model prediction accuracy could be lower in the future. To ensure that the accuracy level is maintained at the target level, the predictive engine must automatically update its quarterly or reduce the classification accuracy level.

3.2. Sampling method

The stratified sampling approach is implemented in this study to ensure that each population subgroup receives proper sample representation. As a result, stratified random sampling provides more extensive population coverage and guarantees that all are included in the sampling. This sampling method also supports current research, which is ticket resolution code prediction, consisting of various groups. Figure 4 displays the resolution code class scatter plot. The resolution code class distribution seems to be disassembled and unbalanced, used by each region. For example, in zone Gombak, the resolution code "TM_CPE_IW_RJ45_Replaced" is applied regularly, and "NTT_resolved" is commonly applied in Zone Puchong.

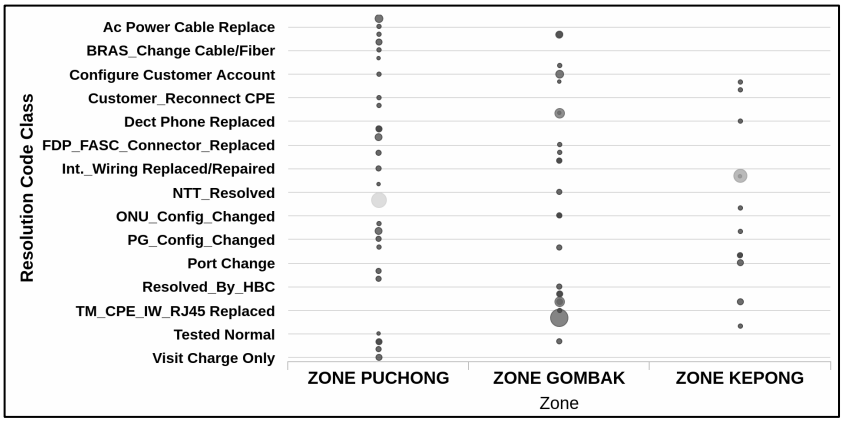

Fig. 4. Resolution code class distribution.

This method also analyzes the class relationship(s) and focuses on a specific category within the class (i.e., resolution code and symptom error code). The stratified sampling is expected to achieve higher accuracy than Simple Random Sampling (SRS) [17]. This situation is due to the smaller variability in the

subgroups. The overall population dataset variance supports current research, which is ticket resolution code prediction, consisting of various groups.

3.3. Automated predictive engine (AutoPE)

The solution to these problems, a framework based on the methods described in earlier parts, called the Automated Predictive Engine (AutoPE), acts as an autonomous module from the trouble ticket system to simplify the building master-detail relationships and predict future resolution code. The resolution code is the final response code within the trouble ticket system. By getting this probability predicted code earlier, engineers and technicians will speed up each fault's troubleshooting guide. Previously, the resolution code is based on the same solution with the same symptom error. This study used the Document-Term-Matrix (DTM) method [18] to enable transposition of the term frequency to become variables for the classification process. To build the DTM tables, AutoPE has access to all past data and gathers synchronous events from the trouble ticket system, including creating a new and closed ticket. If unidentified events occur through a refreshed dataset, AutoPE can update the DTM structure. Figure 5 integration illustrates the methodology of how AutoPE is integrated with the trouble ticket system via a few phases. Technicians can now access the newly created problem ticket for the expected future classification code [19]. Finally, this analytics engine can have functionality on each problem's real scope and must be tested to maintain optimal accuracy with the trained and updated dataset version.

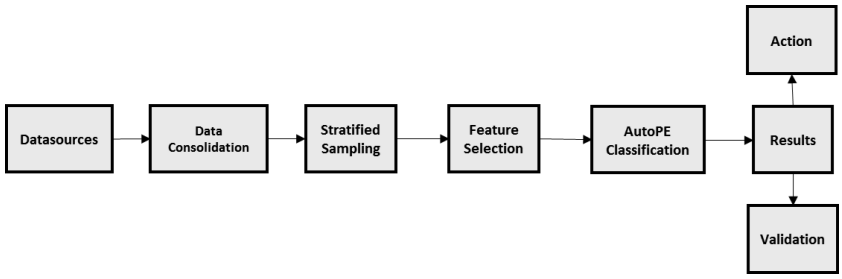

Fig. 5. The methodology of AutoPE development.

This work establishes and implements the classification model at a specific accuracy level. By using the model, the degree of classification accuracy is obtained and periodically observed. The solution is to rebuild the analytical model if the accuracy is below 70%. The classification approach applies only to the same model if it is less than acceptable. The accuracy of over 70% is accepted in this study field due to the large dataset. Due to a few factors, such as the quality of the training data set and the sampling procedure, including stratified random sampling, model accuracy could be lower. Data update frequency is quarterly. Hadoop compiles and merges the new dataset [20]. This rule is designed to ensure that the prediction engine can calculate the accuracy accepted by the current qualified dataset.

This study outlines the initiative to assist technicians and engineers in handling these conditions. The basic concept is to learn the relationship between elements automatically by observing what happened in the past, how network elements interrelate through direct events, or by coincidence through machine learning. This

Natural Language Processing (NLP) [21] related learning method is applied to the vast information gathered over the years to create a vector table for DTM keywords that expresses the likelihood of keywords used in a trouble ticket. These initial vector tables are then modified regularly with current information, as shown in Fig. 6. To build the DTM table, use the R source code below to create the document matrix. The DTM table will be linked to the current dataset, and each document term will become the variables themselves, keeping the word occurrences as each column weights. The different dataset zone will have different column magnitudes, depending on how many keywords each extract.

resolution_code	sum(customer)	sum(customers)	sum(probable) ↓	sum(progress)	sum(process)	sum(semak)
DP_Re-jumper	48	0	5	0	0	0
VDSL_Replaced	145	0	1	5	2	5
Advise Customer	377	0	0	8	18	0
Advise TMUC	101	5	0	1	0	0
AdviseTMUCtoProvide full info	8	0	0	0	0	0
Building power restored	2	0	0	0	0	0
Change Wireless Channel Mode	122	0	0	0	3	0
Configure Customer Account	9	0	0	0	0	0
Configure_STB	1	0	0	0	0	0
Connector_Replaced	1	0	0	0	0	0
Customer_Reconnect CPE	1	0	0	0	0	0

Fig. 6. DTM keyword vector table.

In Fig. 7, line number 91 indicates corporate representation and computing. Corpora is a collection of documents containing text, typically a natural language. A corpus has two metadata types. Corpus metadata contains corpus-specific metadata as tag-value pairs. Document-level metadata contains document-specific metadata but is retained as a corpus data frame. Lines 92 and 94 are a list of features omitted from the corpus. At line 96, the R function "DocumentTermMatrix" constructs or coerces into DTM format. Some additional parameter involves this process, such as removing punctuation, removing all numbers, and converting all characters to lower case for character generalization. Line 104 converts a data frame to a matrix format.

```
90
91 match_corpus = Corpus(VectorSource(df_review$review))
92 stopwords_malay = c("ini","dan","yang","buat","mcm","kat","apa","mmg",
93                     "dah","saya","lagi","dgn","ada","bin","aku","pastu","untuk","perlu")
94 junk_words = c("pun","also","using","actually","amp","can","faz","skrg","lah","abg","nak","kot",
95                "dengan","kepada")
96
97 dtm = DocumentTermMatrix(
98   match_corpus,
99   control = list(
100    removePunctuation = TRUE,
101    stopwords = c(stopwords_malay,stopwords("english"),junk_words),
102    removeNumbers = TRUE, tolower = TRUE
103  )
104 )
105 m <- as.matrix(dtm)
```

Fig. 7. A DTM vector table creation.

First, using R, the instance must start initializing a connection to the H2o components. Some critical parameters need to be passed to the h2o.init feature, such as threads (CPU) and min_mem_size. In this research, the thread input is set to -1, used by the H2o to measure all available CPU cores. For better results, the value for

min_mem_size is set as 12 G.B., the minimum memory acquired for H2o. Figure 8, for example, line 38, acts as the instance shutdown for any H2o instance, and line 39 sets the H2o initialization with unlimited CPU and a minimum 12 G.B. memory. Line 40 is for converting to the H2o data frame from the source of the R data frame.

```
37
38 h2o.shutdown(prompt = F)
39 h2o.init(min_mem_size = "12g", nthreads = -1)
40 df_h20 <- as.h2o(df)
41 dim(df_h20)
```

Fig. 8. H2o initialization using R.

In this analysis, the basic specifications of hardware and software in the following Table 2: -

Table 2. AutoPE hardware and software experimental set.

Hardware	Software
Xeon(R) W-2195 CPU @ 2.30GHz 36 Cores	Ubuntu-based Linux 16.04 (Zorin 12.4)
256 GB DDR4 RAM ECC	Oracle OpenJDK 1.8_0191
500 GB NVMe PCIe M.2 Hard Drive	CRAN (R) version 3.6.3
10 Gbps Network Card	H2o 3.30.0.1 R Library
Hyper-Threading Technology Enable	Cloudera CDH 6.1.0

The algorithms are built on top of the MapReduce framework [22] distributed with H2o and use the Java Fork / Join function for multi-threading by making this customized setup and making H2o core code written in Java. The data is read in parallel and distributed in a compressed way across the cluster and stored in a columnar memory format. These features accelerate the entire classification and prediction process.

3.4. AutoPE variable importance

Variable importance [23] is determined by calculating each variable's relative effect: either the variable to be distributed during the tree building phase, and how large the square error (overall trees) increased (reduced). In this study, a split ratio of 80 (train): 20 (test) with 5-Fold Cross-Validation (CV) [24] was chosen, as implemented in the H2o data collection. Due to the Pareto principles, which make it fairer, this current split ratio is determined, and the dataset also contains high instances and uncertainty on the target variables. Line 51 in Fig. 9 shows a 0.8 ratio of the H2o split function h2o.splitFrame implementation. Separation of the test and training datasets occurs in lines 52 and 53.

```
50 set.seed(123)
51 parts <- h2o.splitFrame(df_h20,ratios = 0.8 , seed = 123)
52 df_train <- parts[[1]]
53 df_test <- parts[[2]]
54 sapply(parts, nrow)
```

Fig. 9. H2o split ratio [80:20] using R.

For example, whenever H2o implements the Distributed Random Forest algorithm, it splits a node based on a numeric or categorical feature. The squared

error reduction assigned is the squared error difference between that node and its child nodes. The calculation assumes the following unbiased estimator below: -

$$VAR = \frac{1}{N}\sum_{i=0}^{N} (y_i - \overline{y})^2 \quad (1)$$

Variance Eq. (1) is the function of a group of values is distributed from the centre of their distribution. The larger the variance, the greater the values. The less, the more conglomerated they are. The arithmetic mean (average) of square deviations from the sample dataset's arithmetic mean is computed as the variance.

$$SE = VAR \times N = [\frac{1}{N} \times \sum_{i=0}^{N} y_i^2 - N \times \overline{y}^2] \times N = [\sum_{i=0}^{N} \frac{y_i^2}{N} - \overline{y}^2] \times N \quad (2)$$

Squared Error Eq. (2) is the difference in squared error between that node and its children's nodes. Figure 10 source code summarized how the Distributed Random Forest (DRF) algorithm implements the variable importance calculation for the selection method above. Some basic model training is required to learn the correct variable to predict the resolution code. The target variable called "*y*" as resolution code, and other columns are combined as "*x*" in line 63.

```
62
63 model_auto <- h2o.randomForest(y = "resolution_code", x= df_columns ,
64                                 training_frame = df_train , seed = 123 )
65 h2o.varimp(model_auto)
66 var_imp <- h2o.varimp(model_auto)
67 print(var_imp)
68 var_imp <- var_imp %>% dplyr::filter(scaled_importance > 0.03)
69 write.csv(var_imp,"var_imp.csv",row.names = F)
```

Fig. 10. Generate variable importance table using R.

The example default seed in this experiment is 123. The seed is pseudo-random number generator initialization, and it is required in this DRF function. Changing the seed is useful for controlling the results with the same values while maintaining randomness and useful for debugging purposes. It also does not determine the classification performance while invoking the function. The h2o.varimp function at line 66 retrieves the list of variables to top-n significance. The variable value is determined by calculating each variable's relative effect: if that variable was selected during the splitting process and how much the Square Error (2) (overall trees) changed. Line 68 is how to filter only variables higher than 0.01. The image below shows the vector significance of Distributed Random Forest (DRF) [25].

In comparison, the raw variable is necessary after being scaled between 0 and 1. H2o shows each element's value. Figure 11 illustrates the value of the numerical scale variable after DRF modelling. Examples of such network-related keywords as "*advancenet*", "*route*", "*ssid*", "*netmask*", and "*supplied*" were inherited from the DTM feature and became one of the significant variables. Each variable is significant, and the distribution was also determined. It also shows the distribution of essential variables within the dataset. High-significance variables are the essential factors, and their values influence the outcome.

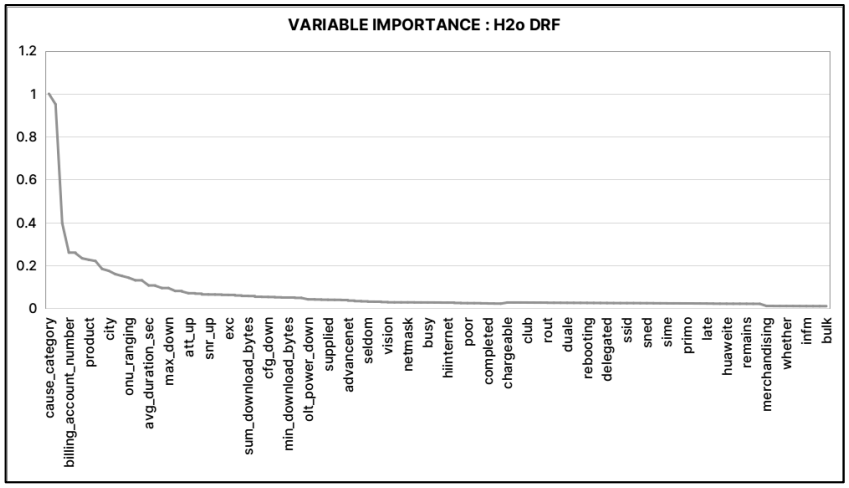

Fig. 11. Example of the DRF variable importance.

3.5. AutoPE Grid (hyperparameter) search

Grid Search is an algorithm-used brute-force technique to train the model and test all possible hyperparameters. This method is an exhaustive analysis of the model's hyperparameter space, either entirely or by a subset. One of the most common ways of doing this is the k-Fold Cross-validation. The hyperparameter tuning strategies can be represented in mathematical formalism by finding the space required for the A procedures. Assume that each dimension of $A \in \mathbf{A}$ is a parameterization of the machine learning pipeline from data processing to model hyperparameter tuning. The process of AutoPE is focused on finding A* (4) to minimize the loss of trained dataset with joined of algorithms and the optimal hyperparameter values.

$$A* = \underset{A \in \mathbf{A}}{\operatorname{argmin}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}\left(A, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}\right) \tag{4}$$

$$\{D_{\text{train}}^{(1)}, \ldots, D_{\text{train}}^{(K)}\} \tag{5}$$

$$\{D_{\text{valid}}^{(1)}, \ldots, D_{\text{valid}}^{(K)}\} \tag{6}$$

$$\mathcal{L}(A, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}) \tag{7}$$

Equations (5) and (6) is the train and validation set acquired by K-Fold CV, and Eq. (7) is the loss for pipeline A data set (i) training and testing data set (i), which evaluated. AutoPE supports two types of grid search: conventional grid search ("*Cartesian*") and random grid search. Users define a set of values for each hyper-parameter they want to look for in a Cartesian grid look. AutoPE will train a model for each hyperparameter value combination. It means that if the model has three hyperparameters and 5, 10, and 2 values are defined for each dimension, and the grid would have a total of (5 × 10 × 2), which equals to 100 models.

The method defines the hyperparameter [26] space, and AutoPE will scan all permutations of hyperparameter values in the range. The process also defines a stop criterion in a grid search, which controls when the search is completed. By defining a maximum number of models or the highest number of times permitted for the

quest, the user may tell the random grid quest to stop. The user may also set a performance-metric-based stop criterion that stops searching for the random grid if the output stops improving by a given amount. Upon completion of the grid search, the user can search the grid object and sort the models by a standard output metric (for example, "*Logloss*", "*RMSE*", "*MSE*"). All models are in the H2o cluster and can be accessed by the name or model I.D. The *h2o.getModel* functionality can be accessed via R code. The model I.D. "*DeepLearning_grid*_1_ *AutoML*_ 20200429_ 235947_ *model*_2" with the naming convention is specified with "*grid*" which means this model has been adjusted to the AutoPE grid Search. During the hyperparameter search regression, the best parameters for this model were specified automatically. Line (10) shows the model I.D. "*GBM*_1_*AutoML*_ 20200429_235947" the Gradient Boosting Machine (GBM) model does not use the Grid Search. This condition could arise due to dimensionality, and it degenerates when determining the number of hyperparameters that expand exponentially. Often there is no guarantee that the *Grid Search* produces the ideal solution, and the Gradient Boosting (GBM) automatically discarded those features.

3.6. AutoPE classification method

The AutoPE object contains a leader board of models trained in the process, including the 5-Fold CV model (default), as shown in Fig. 12. CV is used to evaluate models' predictive performance and evaluate how they perform outside the sample in a new data set, also known as test data. The folds used for the model assessment can be modified using the parameter *nfolds*, and then, the leader board will display scores on that dataset. A default metric ranks the models based on the mean per class error. This study is a multiclass classification problem [27]. The metric means the error per class will be the default matrix.

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
accuracy	0.7420974	0.06717581	0.76666665	0.6516854	0.83146065	0.752809	0.7078652
err	0.25790262	0.06717581	0.23333333	0.3483146	0.16853933	0.24719101	0.29213482
err_count	23.0	5.9581876	21.0	31.0	15.0	22.0	26.0
logloss	2.0338094	0.659052	1.543617	2.9599128	1.3295339	1.9362282	2.3997552
max_per_class_error	1.0	0.0	1.0	1.0	1.0	1.0	1.0
mean_per_class_accuracy	0.8096372	0.051537704	0.82842594	0.77011436	0.8829861	0.8145166	0.75214285
mean_per_class_error	0.19036283	0.051537704	0.17157407	0.22988562	0.11701389	0.18548341	0.24785714
mse	0.2510566	0.069180645	0.21072921	0.3453197	0.16692102	0.24273428	0.28957883
r2	0.9994335	1.7314544E-4	0.9995437	0.9992435	0.9996629	0.99942154	0.999296
rmse	0.49721146	0.06925833	0.45905253	0.5876391	0.4085597	0.4926807	0.5381253

Fig. 12. AutoPE results with cross-validation.

For flexibility also some additional metrics are given, such as Logloss, Root Mean Square Error (RMSE), and Mean Squared Error (MSE), Training Time (*ms*), and Prediction Time per Row (*ms*). Log loss tests a classification model's output where the prediction input is a probability value of 0 to 1. The standard deviation of residuals (prediction errors) is RMSE and MSE is an estimator that calculates the error squares average. Other additional details, such as the training time and prediction time, have also been added. The training time is determined by how long it takes to approximate the relationship between inputs and outputs, i.e., the number of iterations that need to be made to obtain a sufficiently small loss (Logloss). This

process entirely depends on how complex the function is and how useful and noisy the sample data [28] are. Predictive time is more valuable if the time of training is finite, and comparable accuracy is. Figure 13 shows that the model I.D. "*DeepLearning_grid*_1_*AutoML*_ 20200429_235947_*model*_2" is utilizing Deep Learning (DL) algorithm at line 1 is the "leader model" of the AutoPE list and has the lowest Mean per Class Error, which is 0.496, with a log loss of 2.032, RMSE of 0.5, MSE of 0.25, 3633 (*ms*) for training time and 0.112 (*ms*) for prediction time per row. The model I.D. notation, this model of iteration, was explicitly trained using the Deep Learning (DL) algorithm in conjunction with the grid hyperparameter search function to find the model's optimum parameters in anticipation of the highest possible accuracy.

	model_id	mean_per_class_error	logloss	rmse	mse	training_time_ms	predict_time_per_row_ms
1	DeepLearning_grid__1_AutoML_20200429_235947_...	0.4959317	2.032710	0.5009653	0.2509662	3633	0.112022
2	DeepLearning_grid__1_AutoML_20200429_235947_...	0.5047407	1.838764	0.5003037	0.2503038	5696	0.262424
3	DeepLearning_grid__3_AutoML_20200429_235947_...	0.5067180	2.361528	0.5033033	0.2533142	8791	0.335706
4	DeepLearning_grid__1_AutoML_20200429_235947_...	0.5073437	1.744662	0.5003892	0.2503893	10994	0.267151
5	DeepLearning_grid__3_AutoML_20200429_235947_...	0.5075238	2.162241	0.5099426	0.2600415	7477	0.123941
6	DeepLearning_grid__2_AutoML_20200429_235947_...	0.5130817	2.054806	0.5027571	0.2527647	8670	0.341345
7	DeepLearning_grid__2_AutoML_20200429_235947_...	0.5142992	1.967834	0.5082314	0.2582992	9977	0.217771
8	DeepLearning_grid__1_AutoML_20200429_235947_...	0.5161697	1.690254	0.5039302	0.2539456	11496	0.244114
9	DeepLearning_grid__2_AutoML_20200429_235947_...	0.5168397	2.142678	0.5231061	0.2736400	4165	0.080398
10	GBM_1_AutoML_20200429_235947	0.5216192	2.472468	0.5800693	0.3364804	2695	1.629131
11	DeepLearning_grid__1_AutoML_20200429_235947_...	0.5232826	1.625504	0.5027000	0.2527073	11000	0.123661
12	DeepLearning_grid__1_AutoML_20200429_235947_...	0.5256013	1.645192	0.5128055	0.2629694	7914	0.258105
13	DeepLearning_grid__2_AutoML_20200429_235947_...	0.5280630	1.897809	0.5057805	0.2558139	13945	0.164035
14	XGBoost_grid__1_AutoML_20200429_235947_model...	0.5299285	1.467873	0.6171608	0.3808875	9220	0.124048

Fig. 13. Leader board list from AutoPE.

Figure 14 shows a table description that clearly describes some algorithms, such as Deep Learning (DL) [29], GBM [30], Extreme Gradient Boosting (XGBoost) [31], and Extremely Randomized Trees (XRT) [32], which is compatible with this trouble ticket dataset the model numbers produced, which are between 31 to 57. DL base algorithm is selected as the best among the algorithms because of its minimal mean Logloss value, mean RMSE, mean MSE, and smallest mean per class error.

	grp	mean_logloss	mean_rmse	mean_mse	min_mean_per_class_error	min_training_time_ms	min_predict_time_per_row_ms	total_model
1	DL	2.125733	0.5645219	0.3266663	0.4959317	284	0.032688	31
2	DRF	3.268810	0.5838611	0.3408938	0.5334469	998	0.607354	1
3	ENSEMBLE	2.983218	0.8649298	0.7488519	0.8297207	57563	2.037361	2
4	GBM	5.710656	0.7981654	0.6521623	0.5216192	938	0.337988	57
5	GLM	3.265990	0.9396130	0.8828725	0.8833333	40	0.679333	1
6	XGBOOST	2.673650	0.8252272	0.7016626	0.5299285	1977	0.030000	31
7	XRT	4.177543	0.7852017	0.6165416	0.6053865	1391	0.626484	1

Fig. 14. AutoPE supported algorithm.

The selection of the "*leader model*" from the leader board is essential to implement the prediction. The *h2o.getModel* function (line 122) is used to select the desired model and then position it in a variable's name. This variable name will access the model's properties, such as model I.D. (line 123), list of essential variables (line 124), and information about the fold assignment (line 125). More details on the method, as shown in Fig. 15.

```
122 best_model <- h2o.getModel(head(model_auto@leaderboard,1)$model_id)
123 best_model@model_id
124 best_model@model$variable_importances
125 best_model@parameters$fold_assignment
```

Fig. 15. Retrieving for the best model.

Figure 16 explains the H2o method used at (line 134) for *h2o.predict*. The prediction output is compared to the testing dataset, and the output is at (line 135) in the H2o table. For easier manipulation, the table output *h2o.table* is then forced into the R data structure *as.data.frame*. Using the function *h2o.confusionMatrix*, the output matrix is retrieved by forwarding the best model value and the test dataset. Finally, the resulting confusion matrix is created based on the performance value input (line 137-138).

```
134 pred <- h2o.predict(best_model,df_test)
135 pred <- as.data.frame(h2o.table(pred$predict, df_test$resolution_code))
136 perf <- h2o.performance(best_model,df_test)
137 h2o.confusionMatrix(perf)
138 cm <-  as.data.frame(perf@metrics$cm$table)
139 print(cm)
140 eq <- gsub(",","",cm$Rate[nrow(cm)])
141 print(eq)
142 accuracy <- as.data.frame(1-sapply(eq, function(x) eval(parse(text=eq))))
143 colnames(accuracy)[1] <- "value"
144 print(paste0("Classification Accuracy => " , round(accuracy$value,3)*100," %"))
```

Fig. 16. Resolution code prediction using the best model.

The confusion matrix is a matrix that shows the relation in a classification problem between observed and predicted values. In (line 142) is used to obtain the accuracy that derives from the confusion matrix. The R code decodes the string notation into a mathematical equation (*eval*) and calculates the optimal accuracy. The classification accuracy is the proportion of *true positive* predictions divided by the total number of predictions made, multiplied by one hundred to become a percentage (line 144).

4. Results

A simple prototype that applies the techniques defined in this study was implemented by integrating with the existing trouble ticket system that manages many tickets daily. The trouble ticket repository reports anything related to switching, transmission, device, server, and customer-related problems. Tickets provide information on the affected element, information on failure, and management information (ticket identification, a technician in charge, dates, tasks performed). Whenever AutoPE's overall accuracy is lower than the stated threshold, the historical data must be assembled, and the model needs to be rebuilt. The earlier version of the AutoPE module shows that it provides the technicians with critical details to help them overcome the fault (predicted resolution code) and those network elements that may be affected by a specific malfunction, showing the problem's characteristics. To ensure that the study will be useful in the future, the final analysis result was discussed with the team on the field ground and with telco management.

Table 3 shows the results of the classification using AutoPE. Column "*Zone*" is the name of a specific zone group with trouble tickets related to high-speed internet deployment. Column "*Algorithm*" is the classifier list from the AutoPE classification results, "*the leader*". "*Accuracy*" is the percentage of the classification results. At the same time, "*Time*" is the amount of time taken to complete the classification process, the "*Model*" column is the number of the model created by the selected classifier, "*Grid*" column is the labelling for that classifier using the Grid Search method.

Table 3. AutoPE resolution code classification accuracy.

Zone Name	Algorithm	Accuracy (%)	Time (Hours)	Total Model	Grid	Total Sample
BANGI	GBM	72.5	0.5877	44	YES	2568
BANGSAR	DL	63.6	0.5620	107	YES	1581
B. ANGGERIK	DL	73.8	0.5990	235	YES	1515
CYBERJAYA	GBM	75.0	0.5857	183	YES	3024
GOMBAK	DL	75.6	0.5629	93	YES	3519
KPG BATU	DL	81.9	1.252	29	YES	2377
KERAMAT	DL	**86.6**	1.114	33	YES	3306
KLANG	DL	70.1	0.5572	42	YES	2154
MALURI	XGBOOST	72.4	0.5740	28	YES	1341
PANDAN	GBM	77.3	0.5494	132	YES	1702
PUCHONG	DL	**66.7**	0.5691	95	YES	1986
SBG JAYA	DL	69.3	0.5806	52	YES	1550
S.A. BANTING	DL	79.5	1.127	35	YES	3623
TAR	DL	77.5	1.0893	83	YES	1498
TAMPOI	XGBOOST	73.3	1.4894	176	YES	1777
T. PETALING	DL	74.3	0.5816	34	YES	1209
TDI	DL	80.4	1.108	28	YES	3623
ALL ZONE	DRF	**79.5**	2.418	10	YES	38353

The table also reveals that the highest classification results are from Zone Keramat, which scores 86.6% accuracy, and the lowest score is 66.7% from Zone Puchong. The cause of the lower accuracy classification results may be that the sampling size is not large enough, or the sample dataset might contain noisy characteristics [19], and it may also not have sufficient variations in the dataset. There will be some investigations to determine the root cause, particularly in this zone. Finally, this study also attempts to predict the resolution code for the entire dataset (all zone) sizing of 38,353 records. The prediction accuracy scores 79.5% using the DRF classifier without the Grid Search, indicating that this dataset fits the classification method.

Without any hyperparameter tuning assistance, Table 4 shows the classification results using the traditional method of ML. Most of the accuracy is over 65 %, and the type of algorithm for each zone is selected randomly. The highest accuracy score is from the Keramat zone, which is 77.2%, and the lowest score is 54.5% accuracy from the Bangsar zone. Many of the algorithm scores are very low from the time taken for classification, which means that it took very minimal time to complete the classification process. The main reason is that the traditional method generated only one model per algorithm, and no hyperparameter tuning was involved. The entire zone classification system chosen for each compatible algorithm, such as GBM, DRF, DL, and XGBoost, is ideal for benchmarking the overall output for the entire 38,353 records. The standard DRF algorithm's performance is almost the same as the AutoPE results (DRF, 79.5 %), 79.4 %, and

uses no hyperparameter tuning (Grid = No). Thus, the DRF algorithm is selected for the traditional classification method.

Table 4. Traditional ML (DRF) resolution code classification accuracy.

Zone Name	Algorithm	Accuracy (%)	Time (Hours)	Total Model	Grid	Total Sample
BANGI	DRF	64.7	0.0137	1	NO	2568
BANGSAR	DRF	**54.5**	0.0166	1	NO	1581
B. ANGGERIK	DRF	66.7	0.0133	1	NO	1515
CYBERJAYA	DRF	66.7	0.0211	1	NO	3024
GOMBAK	DRF	65.1	0.0148	1	NO	3519
KP BATU	DRF	59.1	0.0662	1	NO	2377
KERAMAT	DRF	**77.2**	0.0220	1	NO	3306
KLANG	DRF	62.1	0.0568	1	NO	2154
MALURI	DRF	64.5	0.0118	1	NO	1341
PANDAN	DRF	72.7	0.0568	1	NO	1702
PUCHONG	DRF	60.6	0.0154	1	NO	1986
SBG. JAYA	DRF	61.4	0.0281	1	NO	1550
S.A BANTING	DRF	75.8	0.0338	1	NO	3623
TAR	DRF	66.2	0.0108	1	NO	1498
TAMPOI	DRF	60.0	0.0066	1	NO	1777
T. PETALING	DRF	69.5	0.0138	1	NO	1209
TDI	DRF	67.6	0.0280	1	NO	3623
ALL ZONE	DRF	**75.4**	0.2278	1	NO	38353

In conclusion, some algorithms like DRF sometimes do not require hyperparameter once they achieve some optimal accuracy threshold, and the dataset is fit with the algorithm. The DRF will also automatically extend the tree's right amount to find the dataset's right split. The more deeply the tree is split, the more knowledge on the data is obtained. Figure 17 shows in graphical classification performance comparison between AutoPE and the traditional ML method (accuracy and time), while Figure 17 illustrates how significant AutoPE can boost classification results. Another additional analysis is through the study of the association between the tests. The aim of performing the correlation hypothesis test's significance is to determine if the sample data's linear association is sufficiently high to model the population's relationship and potential improvements.

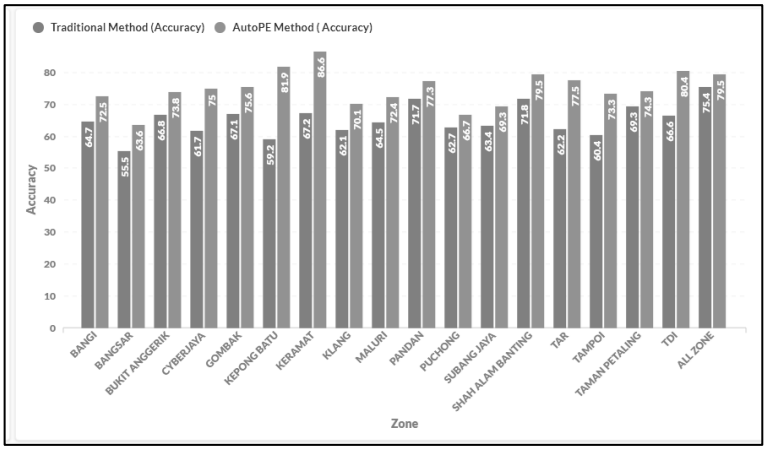

Fig. 17. AutoPE vs traditional method ML classification (accuracy).

5. Conclusions

When network structures are complex involving thousands of different types of interconnected components, it takes considerable effort to locate the hidden association. It needs trained technicians to conduct several tests to verify causal failure. This method is time-consuming based on engineers' and technicians' expertise and experience. Integrating AutoPE with the trouble ticket system will synchronize all the information gathered over the years and recognize the network components' fault relationships. This approach will warn unknown root causes of failures, minimize downtime, increase customer satisfaction, and reduce maintenance costs. It can also measure the severity of the current fault, impact network components, early warning of failures, prevent significant failure events, and provide a more reliable estimation of any potential loss. The Mean Time to Recovery (MTTR) and the effect of failure on customer services will reduce significantly.

The results show that the implementation with AutoPE improves classification efficiency range from 5% to 38% compared to the previous traditional method. One significant advantage of applying the AutoPE on the trouble ticket dataset is that the technique addressed the existing limitation of traditional machine learning. It is ready to blend in the enterprise telco environment that requires repetitive and extensive work of classification due to the trouble tickets data's velocity and at the same time to maintain the best accuracy as possible.

One of the main drawbacks found in this method is having high quality and consistency labelled data to maintain accuracy. Future works in this study will integrate some additional features, such as integration with the big data computing platform, including MapReduce and Spark. This integration enables automated machine learning to use the full Hadoop unified in-memory large-scale data processing engine. This integration also addresses drawbacks, including memory size issues and processing power. The application of using the AutoPE will be extended for other telco-related use cases such as churn prediction, marketing campaign classification, and customer behaviour segmentation that require tremendous effort to analyse due to the enormous size and high velocity of the dataset.

Nomenclatures

$D_{\text{train}}^{(1)}$	Train dataset
$D_{\text{train}}^{(K)}$	K numbers of train dataset
$\mathcal{L}(A, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)})$	Logarithmic loss value for train and validation dataset
SE	Squared Error
VAR	Variance

Greek Symbols

$\in$	Element of symbol

Abbreviations

A.I.	Artificial Intelligence
AutoML	Automated Machine Learning
AutoPE	Automated Predictive Engine

CIB	Customer Internet Bandwidth
CISQ	Customer Internet Service Quality
CP	Customer Profiles
CRAN	Comprehensive R Archive Network
CTT	Customer Trouble Ticket
CV	Cross-Validated Method
DL	Deep Learning
DRF	Distributed Random Forest
DTM	Document Term Matrix
G.B.	Gigabyte
GBM	Gradient Boosting Machine
HDFS	Hadoop Distributed File System
IT	Information Technology
k-Fold	K number of Fold (Split) Sampling
ML	Machine Learning
MSE	Mean Squared Error
MTTR	Mean Time To Recovery
NLP	Natural Language Processing
RMSE	Root Mean Square Error
XGBoost	Extreme Gradient Boosting
XRT	Extremely Randomized Trees

References

1. Molinero Velasco,; and Fco.J. (2012). A Bayesian Network approach to diagnosing the root cause of failure from Trouble Tickets. Artificial Intelligence Research, 1(2), 75. https://doi.org/10.5430/air.v1n2p75
2. Marlow, D.W. (2004). Investigating technical trouble tickets: an analysis of a homely CMC genre. *Proceedings of the* 37*th Annual Hawaii International Conference on System Sciences*, Big Island, USA.
3. Weng, Z. (2019). From conventional machine learning to AutoML. *Journal of Physics: Conference Series*, 1207(1), 1-10.
4. Zheng, J.; and Dagnino, A. (2014). An initial study of predictive machine learning analytics on large volumes of historical data for power system applications. *Proceedings of the* 2014 *IEEE International Conference on Big Data* (*Big Data*). Washington, USA, 952-959.
5. Asres, M.W.; Mengistu, M.A.; Castrogiovanni, P.; Bottaccioli, L.; Macii, E.; Patti, E.; and Acquaviva, A. (2020). Supporting telecommunication alarm management system with trouble ticket prediction. *IEEE Transactions on Industrial Informatics*, 17(2), 1459-1469.
6. Symonenko, S.; Rowe, S.; and Liddy, E.D. (2006). Illuminating trouble tickets with sublanguage theory. *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA, 169-172.
7. Temprado, Y.; Molinero, F.J.; Garcia, C.; and Gomez, J. (2008). Knowledge discovery from trouble ticketing reports in a large telecommunication company. *Proceedings of the* 2008 *International Conference on Computational Intelligence for Modelling Control and Automation*. Vienna, Austria, 37-42.

8. Medem, A.; Akodjenou, M.-I.; and Teixeira, R. (2009). TroubleMiner: mining network trouble tickets. *Proceedings of the* 2009 *IFIP/IEEE International Symposium on Integrated Network Management-Workshops*. New York, USA, 113-119.
9. Potharaju, R.; Jain, N.; and Nita-Rotaru, C. (2013). Juggling the jigsaw: towards automated problem inference from network trouble tickets. *Proceedings of the* 10*th USENIX Symposium on Networked Systems Design and Implementation*. Lombard, Illinois.
10. Tan, J.S.; Ho, C.K.; Lim, A.H.-L.; and Ramly, M.R.M. (2018). Predicting network faults using random forest and C5.0. *International Journal of Engineering and Technology*, 7(2), 93-96.
11. Jhamb, D.; Mittal, A.; and Sharma, P. (2020). The behavioural consequences of perceived service quality: a study of the indian telecommunication industry. *Business: Theory and Practice*, 21(1), 360-372.
12. Teo, T.S.H. (2001). Demographic and motivation variables associated with Internet usage activities. *Internet Research: Electronic Networking Applications and Policy*, 11(2), 125-137.
13. Tholath, D.I.; and Casimirraj, S.J. (2016). Customer journey maps for demographic online customer profiles. *International Journal of Virtual Communities and Social Networking*, 8(1), 1-18.
14. Kune, R.; Konugurthi, P.K.; Agarwal, A.; Rao, C.R.; and Buyya, R. (2016). XHAMI - extended HDFS and MapReduce interface for Big Data image processing applications in cloud computing environments. *Software Practice and Experience*, 47(3), 455-472.
15. Mehta, S.; and Mehta, V. (2016). Hadoop ecosystem: an introduction. *International Journal of Science and Research*, 5(6), 557-562.
16. Levin, R.; and Kanza, Y. (2014). Stratified-sampling over social networks using mapreduce. *Proceedings of the* 2014 *ACM SIGMOD International Conference on Management of Data*. Snowbird Utah, USA, 863-874.
17. Alim, A.; and Shukla, D. (2019). An application approach of stratified sampling in analytic-predictive environments of big data. *Proceedings of the International Conference on Sustainable Computing in Science, Technology and Management*. Jaipur, India.
18. Raies, A.; Mansour, H.; Incitti, R.; and Bajic, V.B. (2013). Combining position weight matrices and document-term matrix for efficient extraction of associations of methylated genes and diseases from free text. *PLoS ONE*, 8(10).
19. Pikies, M.; and Ali, J. (2019). String similarity algorithms for a ticket classification system. *Proceedings of the* 2019 6*th International Conference on Control, Decision and Information Technologies*. Paris, France, 36-41.
20. Laxmi, Y.S.; Mahendar, A.; and Gouse, S. (2017). Customer complaint analysis using hadoop (consumer analysis). *International Journal of Engineering and Computer Science*, 6(4).
21. Menon, S.; and Nagalakshmi, M. (2020). Techniques of NLP that improve effectiveness in an english language classroom. *International Journal of Psychosocial Rehabilitation*, 24(2), 1569-1578.
22. Ptiček, M.; and Vrdoljak, B. (2017). MapReduce research on warehousing of big data. (2018). *Proceedings of the* 40*th International Convention on*

Information and Communication Technology, Electronics and Microelectronics. Opatija, Croatia, 1-6.

23. Tadist, K.; Najah, S.; Nikolov, N.S.; Mrabti, F.; and Zahi, A. (2019). Feature selection methods and genomic big data: a systematic review. *Journal of Big Data*, 6(1).
24. Okfalisa; Gazalba, I.; Mustakim; and Reza, N.G.I. (2017). Comparative analysis of k-nearest neighbor and modified k-nearest neighbor algorithm for data classification. *Proceedings of the* 2*nd International Conferences on Information Technology, Information Systems and Electrical Engineering*. Yogyakarta, Indonesia, 294-298.
25. Herrera, V.M.; Khoshgoftaar, T.M.; Villanustre, F.; and Furht, B. (2019). Random forest implementation and optimization for big data analytics on LexisNexis's high performance computing cluster platform. *Journal of Big Data*, 6(1).
26. Kadam, V.J.; and Jadhav, S.M. (2020). Performance analysis of hyperparameter optimization methods for ensemble learning with small and medium sized medical datasets. *Journal of Discrete Mathematical Sciences and Cryptography*, 23(1), 115-123.
27. Ndirangu, D.; Mwangi, W.; and Nderu, L. (2019). Using ensemble technique to improve multiclass classification. *Journal of Information Engineering and Applications*, 9(5), 28-42.
28. Kreek, R.A.; and Apostolova, E. (2018). Training and prediction data discrepancies: challenges of text classification with noisy, historical data. *Proceedings of the* 2018 *EMNLP Workshop W-NUT: The* 4*th Workshop on Noisy User-Generated Text*. Brussels, Belgium, 104-109.
29. Mackey, T.K.; Li, J.; Purushothaman, V.; Nali, M.; Shah, N.; Bardier, C.; Cai, M.; and Liang, B. (2020). Big data, natural language processing, and deep learning to detect and characterize illicit COVID-19 product sales: infoveillance study on twitter and instagram. JMIR Public Health and Surveillance, 6(3).
30. Flores, V.; and Keith, B. (2019). Gradient boosted trees predictive models for surface roughness in high-speed milling in the steel and aluminum metalworking industry. *Complexity in Manufacturing Processes and Systems*, 2019, 1-15.
31. Wang, M.-X.; Huang, D.; Wang, G.; and Li, D.-Q. (2020). SS-XGBoost: a machine learning framework for predicting newmark sliding displacements of slopes. *Journal of Geotechnical and Geoenvironmental Engineering*, 146(9), 1-17.
32. Saeed, U.; Jan, S.U.; Lee, Y.-D.; and Koo, I. (2021). Fault diagnosis based on extremely randomized trees in wireless sensor networks. *Reliability Engineering and System Safety*, 205.