Distributed Data Mining for Root Causes of KPI Faults in Wireless Networks

Shiliang Fan[1], Yubin Yang[1,*], Wenyang Lu[1], and Ping Song[2]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
yangyubin@nju.edu.cn
[2]Huawei Technologies Co.Ltd, Shanghai, China
songping@huawei.com

Abstract. In the field of wireless network optimization, with the enlargement of network size and the complication of network structure, traditional processing methods cannot effectively identify the causes of network faults in the face of increasing network data. In this paper, we propose a root-cause-analysis method based on distributed data mining (DRCA). Firstly, we put forward an improved decision tree, where the selection of the best split-feature is based on the feature's purity-gain, and then we skillfully convert the problem of root-cause-analysis into modeling of an improved decision tree and interpretation of the tree model. In order to solve the problem of memory and efficiency associated with large-scale data, we parallelize the algorithm and distribute the tasks to multiple computers. The experiments show that DRCA is an effective, efficient, and scalable method.

Keywords: KPI Faults, Root-Cause-Analysis, Improved Decision Tree, Distributed Data Mining, Parallelization Algorithm

1 Introduction

Wireless network optimization is an important part of the network operation process [2]. In general, the monitoring of network performance relies on a predefined set of key performance indicators (KPIs). Faults in these KPIs always represent the deterioration of network performance [2].

The network system outputs numerous KPI reports daily. Engineers traditionally analyze these reports to identify the specific causes of KPI faults based on accumulated experience and data processing tools [1,7,8]. However, the enlargement of network size and the complication of network structure leads to inaccuracies and inefficiencies with the traditional method. There are two issues pertaining to KPI faults in wireless networks: firstly, the KPI reports for analysis lack root causes labels, so the problems cannot be transformed into a multi-classification of the root causes; secondly, with the limitation of RAM capacity and CPU speed [11], it is difficult to obtain timely and effective results when dealing with large-scale KPI reports.

2 Problem Definition and Analysis

Each row of the report represents one sample. Assuming the KPI report has N(N is very large) samples in total, expressed as $x_i(1 \leqslant i \leqslant N)$, each column represents a feature of x_i. The structure of the report is as shown in Table 1.

Table 1. The structure of the KPI report

Index	Start Time	Period	Cell	Indicator_1	$\cdots$	$\textbf{Indicator}_\textbf{K}\textbf{(KPI)}$	Indicator_M
1							
$\cdots$							
N							

We define a set S containing all $Indicators$ from $Indicator_1$ to $Indicator_N$, with the exception of $\boldsymbol{Indicator_K(KPI)}$. To determine the set of $\{Root\_Cause_i\}$ ($Root\_Cause_i \in S$) leading to the KPI fault, and the corresponding set of $\{Weight_i\}$($Weight_i$ represents the influence of $Root\_Cause_i$ on the KPI fault), we take a series of indicators in the set S as features, and the occurrence of KPI($Indicator_K$) fault as labels to train the binary classification model. Then, we interpret the model to obtain each root cause and its weight. Based on the considerations of interpretability and parallelizability, we have selected the decision tree [10] as the classification model.

3 Method Introduction

3.1 Data Preparation

We regard whether the KPI fault occurs as the training target (label), so it is necessary to extract the KPI of each x_i and compare the value with the industry threshold. If the KPI fault occurs, then $label_i$ = -1 otherwise $label_i = 1$.

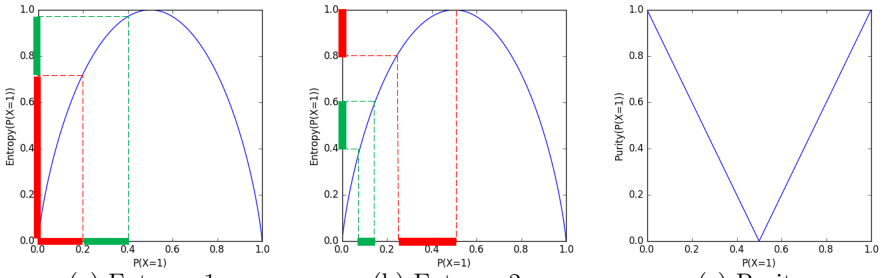
3.2 Model Training

Theoretical Principle. Based on the idea of the decision tree [9,10] we view "*KPI fault occurrence*" as the classification target. In the growth of the decision tree, we continuously select $Indicator_i(Indicator_i \in S)$ as the decision-making basis. If $Indicator_i$ can provide more information for classification, it is more likely to be a root cause.

However, it is clear that our real intention of the training decision tree is not to classify, but to obtain the extent of effective information brought by each indicator for classification. As shown in Fig.1(a) and Fig.1(b), there is a non-linear relationship between information entropy and the distribution of a variable X in binary classification. Therefore, there will be some deviation if we use entropy-gain to quantify the influence of each feature on classification.

Therefore, we propose another criterion to select features when training the decision tree. Assuming the proportion of negative samples whose $label$ = -1 in data-set D is p^-, we define the **purity** of D as follows:

$$Purity(D) = |2 * p^- - 1| \,. \tag{1}$$



(a) Entropy-1 (b) Entropy-2 (c) Purity

Fig. 1. The relationship between the distribution and entropy/purity.

As shown in Fig.1(c), there is linear relationship between purity and the distribution of the variable X in binary classification. Since all indicators are continuous, we use the dichotomy to process these continuous features. Suppose that continuous feature a has L different values on D, recording as $\{a_1, a_2, \ldots, a_L\}$ after ranking in ascending order. The data-set D can be divided into subsets D_t^- and D_t^+ by split-point t, where D_t^- contains the samples with values on feature a not greater than t, and D_t^+ the samples with values greater than t. We adopt directly each $a_i (1 \leqslant i \leqslant L)$ forming the set of alternative split-points:

$$T_a = \{a_i | 1 \leqslant i \leqslant L\} \ . \tag{2}$$

Based on the definition of purity, we define the **purity-gain** similarly to the information entropy-gain:

$$\begin{aligned} Gain(D, a) &= \max_{t \in T_a} \ Gain(D, a, t) \\ &= \max_{t \in T_a} \ \sum_{\lambda \in \{-,+\}} \frac{|D_t^\lambda|}{|D|} Purity(D_t^\lambda) - Purity(D) \ . \end{aligned} \tag{3}$$

Concrete Implementation. To calculate the purity-gain of each feature, we gather their information in parallel using MapReduce. The breadth-first strategy is used to generate the decision tree layer by layer so that with a single execution, the statistical information of all tree-nodes on the same layer can be obtained simultaneously.

In order to record the position of each tree-node and to divide the data-set into the tree-nodes expediently, we design the following data structure:

$$\begin{array}{l} Node: \\ \quad Path: Map<indicator\_id, Pair<indicator\_value, comparison>> \\ \quad Label: String \end{array}$$

where $Path$ represents the series of conditions that should be met from the root-node to the current $Node$. $Label$ can be "$inside$" or "$leaf$" which means $Node$ is an internal node or a leaf node.

In order to avoid excessive growth of the decision tree, we set a parameter called *limit*. When the purity-gain of the best split-feature on a node is less than *limit*, the node will stop splitting and be marked as a leaf node.

The training process of a model is described as Algorithm 1.

Algorithm 1: DecisionTreeDriver

```
   Input: DPath: file path of the data-set after data preparation
   Output: Tree: the trained model of decision tree
 1 currentQueue ← new LinkedList<Node> ;
 2 rootNode ← new Node() ;
 3 Insert rootNode to currentQueue ;
 4 while currentQueue is not empty do
 5     newQueue ← new LinkedList<Node> ;
 6     write currentQueue to HDFS ;
 7     ProduceStatisticalInfo(DPath) ; //run MapReduce Job to do statistics on data-set
 8     for each node ∈ currentQueue do
 9         read the statistical information from HDFS ;
10         bestPG , bestFeatureID , bestSplitPoint ← SelectBestSplitFea(node) ;
11         if bestPG < limit then
12             node.Label ← "leaf"
13         else
14             node.Label ← "inside" ;
15             leftPath ← node.Path.add(bestFeatureID,Pair<bestSplitPoint,0>) ;
16             leftNode ← new Node(leftPath) ;
17             rightPath ← node.Path.add(bestFeatureID,Pair<bestSplitPoint,1>) ;
18             rightNode ← new Node(rightPath) ;
19             insert leftNode,rightNode to newQueue ;
20         add node to Tree
21     currentQueue ← newQueue ;
22 return Tree
```

The function $ProduceStatisticalInfo$ represents statistics on large-scale samples with MapReduce. In the phase of Map, for each $Node_j$, if the sample x_i meets the $Path$ of $Node_j$ then a key-value pair recording as $<key, value>$ will be output. The Reducers receive the pairs of $<key, value>$ output by Mappers and add the values of the same key up recording as $value\_sum$. The $value\_sum$ is actually the number of samples which meet the key. Then write all pairs of $<key, value\_sum>$ to HDFS.

$SelectBestSplitFea$, as the name suggests, is the function to select the best split-feature of each node according to the output above produced by MapReduce. With the help of the characteristic that all pairs of $<key, value\_sum>$ are sorted by key and no key is repeated after the Reduce function, we only need to traverse the output once to determine the purity-gain of each feature.

3.3 Reverse Interpretation

After training the model, suppose that the decision tree has T nodes numbered $1, 2, \ldots, T$. We use an array named $node\_fea$ to record the best split-feature of each node while another named $node\_pg$ records the purity-gain of the above-mentioned split-feature. We make use of MapReduce to divide all samples into the corresponding paths of the decision tree in parallel, and then record the number of negative samples passing through each tree-node in the array *statistic*.

In the first instance, we acquire the indexes of tree-nodes with the same best split-features, for each $Indicator_i (Indicator_i \in S)$ expressed as ind_i:

$$Indexes(ind_i) = \{index | node\_fea[index] = ind_i (ind_i \in S)\} . \quad (4)$$

Since the same split-feature may appear in different nodes, we define the weighting purity-gain for each feature as follows:

$$P\_Gain(ind_i) = \sum_{j \in Index(ind_i)} \frac{statistic[j] \times node\_pg[j]}{\sum_{k \in Index(ind_i)} statistic[k]} . \quad (5)$$

Considering that the number of samples divided by different split-features is also different while training the model, the absolute weight of ind_i is defined as:

$$Abs\_Weight(ind_i) = \frac{P\_Gain(ind_i) \times \sum_{j \in Index(ind_i)} statistic[j]}{\sum_{k=1}^{T} statistic[k]} \times 100\% . \quad (6)$$

Then the relative weight of ind_i is obtained:

$$Rela\_Weight(ind_i) = \frac{Abs\_Weight(ind_i)}{\sum_{ind_i \in S} Abs\_Weight(ind_i)} \times 100\% . \quad (7)$$

4 Experiment and Analysis

In order to verify the effect of **DRCA** in mining the root cause of KPI faults, we conduct a series of relevant experiments and validate the effectiveness and efficiency of this method.

4.1 Experimental Environment

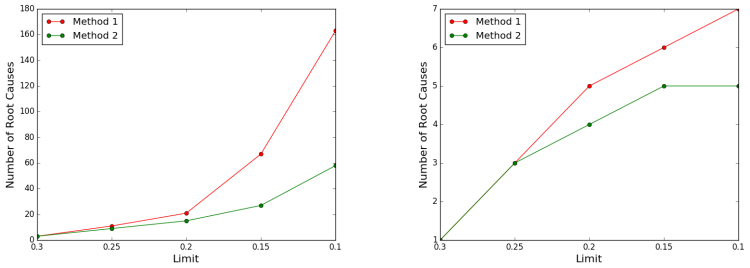
The data-set used in this study is provided by a company in the field of communication and sampled from a realistic communication network. The data-set contains 2203740 samples and the composition of each sample has been explained previously. Excluding *Index*, *StartTime*, *Period*, and *Cell* there are 57 indicators in the data-set, including 5 KPIs. The remaining 52 indicators may possibly lead to KPI faults. Furthermore, we take a Hadoop cluster with 1 master and 4 slaves to do experiments.

4.2 Criteria

Regarding the problem raised in this paper, it's difficult to do an experiment compared with other conventional methods. On one hand, those methods can't output the weight of each root-cause; On the other hand, due to the lack of sufficient labeled data as a criterion, we can't quantify the accuracy of the result mined by each method. To verify the effectiveness of the model, we propose the following reliability criteria, relying on the analysis of the problem combined with the real situation:
(1)The root-causes obtained should be relatively stable and concentrated;
(2)The root-causes obtained should coincide partly with the manual experience.
Additionally, the efficiency is evaluated by comparing with the serial algorithm.

4.3 Effectiveness Experiment

In the experiment, we focus on the KPI called "*RRC Setup Success Rate*". According to the industry standard, this KPI breaks down when its value is less than 99.5%. We mine the possible root-causes which lead to the fault of it from the set S. When training the decision tree model, we change the feature selection criterion from entropy-gain (Method 1) to purity-gain (Method 2). Then, we compare the results of the two methods under the different values of *limit*.



(a) Number of tree nodes (b) Number of root causes

Fig. 2. The mining results of two different methods.

It can be seen from Fig.2(a) that, with the limit decreasing, the growth rate of the decision tree trained by Method 1 is significantly higher than that of Method 2. As shown in Fig.2(b), there is not a significant difference in the number of root causes between the two methods. However, further analysis of Table 2 shows that the root causes resulting from Method 1 are not stable and the weight of fractional root causes changes significantly. By comparison, the root causes obtained by Method 2 are considered to be stable.

Table 2. The root-causes and their weight of two methods

Limit	Method 1		Method 2	
	Root Cause	Weight	Root Cause	Weight
0.3	L.ChMeas.PDCCH.AggLvl8Num	100.00%	L.ChMeas.PDCCH.AggLvl8Num	100.00%
0.25	L.ChMeas.PDCCH.SymNum.3	34.93%	L.ChMeas.PDCCH.SymNum.3	44.27%
	L.ChMeas.PDCCH.AggLvl8Num	31.46%	L.ChMeas.PDCCH.AggLvl8Num	32.15%
	L.ChMeas.PDCCH.SymNum.2	27.67%	L.ChMeas.PDCCH.SymNum.2	23.58%
0.15	L.ChMeas.PDCCH.SymNum.2	22.74%	L.ChMeas.PDCCH.SymNum.3	30.71%
	L.ChMeas.PDCCH.SymNum.3	19.25%	L.ChMeas.PDCCH.AggLvl8Num	22.31%
	L.ChMeas.PDCCH.AggLvl8Num	16.21%	L.ChMeas.PDCCH.SymNum.2	14.33%
	L.ChMeas.CCE.Avail	14.69%	L.ChMeas.CCE.Avail	16.22%
	L.ChMeas.PDCCH.SymNum.1	13.57%	RRC ConnReq attempt	11.06%
	RRC ConnReq attempt	10.34%		

Drawing on the experience provided by the above company, the main reasons for the fault of "*RRC Setup Success Rate*" are the following: weak coverage, strong interference, and channel congestion. The root causes mined by Method 2 mostly point to the "*Physical Down link Control Channel*" (PDCCH), which is perceptibly consistent with the manual experience — channel congestion.

Considering the stability, concentration, and consistency with manual experience of the root causes resulting from Method 2, it can be proved that the method we propose is effective.

4.4 Efficiency Experiment

Simultaneously, we achieve the serial version of our method and compare it with DRCA. The following experiments deal with the KPI named "*RRC Setup Success Rate*", and are conducted on the basis of $limit = 0.15$.

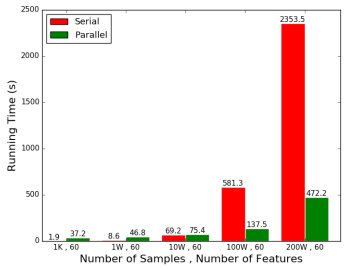


Fig. 3. The running time of two methods.

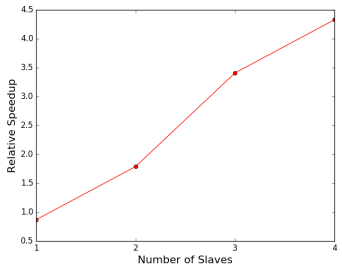


Fig. 4. The relative speedup of DRCA.

The 1K, 10K, 100K, 1M, and 2M samples are selected from the original data-set as the experimental data, and the running time of serial algorithm is compared with that of the DRCA on different data. As can be seen from Fig.3, in terms of small-scale data, the parallel algorithm is actually not as efficient as the serial. With the increase in the number of samples, the efficiency of the stand-alone serial algorithm decreases while DRCA performs well, which shows that DRCA can greatly improve the efficiency in the face of large-scale data.

Then, we test the relative speedup of DRCA with the 1M samples by changing the number of slave nodes in the cluster. As shown in Fig.4, with the increase in the number of slaves in the cluster, the computing capability of DRCA is enhanced, which proves that the algorithm is well scalable.

5 Related Work

In root-cause-analysis and fault detection, there are many researches based on machine learning, including supervised models such as logistic regression [5] and k-nearest neighbor [6], as well as unsupervised models such as clustering [3]. However, these models [3,4,5,6] are not suitable for our research, as we require not only the root-causes of KPI faults, but also the weight of each root-cause. Thus the model must have strong interpretability in order to interpret the answer from the trained model, while the models mentioned above do not meet these requirements. Further, facing large-scale data, the stand-alone models mentioned above cannot better solve the bottlenecks of memory and computing speed.

6 Conclusions

In this study, first we analyze the difficulties in identifying the root-causes of KPI faults with the manual method. Then, we propose a DRCA, which draws

the concept of purity and purity-gain from the main idea of the decision tree, and constructs an improved decision tree using purity-gain as the selection criterion of the best split-feature. Finally, we obtain the suspicious root-causes and their weight by means of reverse interpretation. We achieve the parallel algorithm to execute it on Hadoop. Experiments show that this method can overcome the drawbacks of the traditional methods. Facing massive data, it ensures the effectiveness and also takes the efficiency and scalability into consideration. Obviously, this method has certain value and instructional significance in the practical optimization scene of wireless networks. What's more, it's a new method of root-cause-analysis, which can adopt to most data-sets with requirements of root-cause mining, including big data.

Acknowledgement

This work is sponsored by Huawei Innovation Research Program(HIRP Grant No. YB2015090007), the Natural Science Foundation of China (Grant Nos. 61673204, 61321491), the Program for Distinguished Talents of Jiangsu Province, China (Grant No. 2013-XXRJ-018), and the Fundamental Research Funds for the Central Universities (Grant No. 020214380026).

References

1. Method and system for optimising the performance of a network, `https://www.google.com/patents/US20040266442`
2. Ye Ouyang, M. Hosein Fallah: A Performance Analysis for UMTS Packet Switched Network Based on Multivariate KPIS. In: IJNGN, pp. 80–92 (2010)
3. Klaus Julisch: Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security. 6(4), 443–471 (2003)
4. Mike Chen, Alice X. Zheng, Jim Lloyd, Michael I. Jordan, Eric Brewer: Failure diagnosis using decision trees. In: ICAC, pp. 36–43 (2004)
5. Diana Lpez-Soto, Soumaya Yacout, Francisco Angel-Bello: Root cause analysis of familiarity biases in classification of inventory items based on logical patterns recognition. Computers & Industrial Engineering. 93, 121–130 (2016)
6. Margret Bauer, John W. Cox, Michelle H. Caveness, James J. Downs, Nina F: Nearest Neighbors Methods for Root Cause Analysis of Plantwide Disturbances. In: Ind.Eng.Chem.Res, pp. 5977–5984 (2007)
7. Mark Doggett: A statistical comparison of three root cause analysis tools. Journal of Industrial Technology. 20(2), 2–9 (2004)
8. Mark Doggett: Root Cause Analysis: A Framework for Tool Selection. The Quality Management Journal. 12(4), 34–45 (2005)
9. N. Kwad, C. H. Choi: Input feature selection for classification problem. IEEE Trans on Neural Networks. 13(1), 143–159 (2002)
10. J. R. Quinlan: Induction of decision trees: Machine Learning (1984)
11. Girija J. Narlikar: A parallel, multithreaded decision tree builder. Technical report, Carnegie Mellon University (1998)