Knowledge And Rule Based Learning Engine To Analyse The Logs For Troubleshooting.

Prashant Achari[1] Susanta Adhikary[2] Jayashree Madugundu[3] Mungara Jitendranath[4]
[2]CS Engineering, [3]Assistant Professor [4]Professor & Dean,
[1, 3, 4] Dept. of ISE, R V College of Engineering, Karnataka, India
[2]HP India Software Optns, India

Abstract--- Technical support for software or hardware is always a challenge, both from the complexity and investment standpoint. Today the dependency on computers, networked systems and managing applications are becoming more dominant and only log files are available to trace the applications. Log file analysis process is heavily involved, in software, hardware as well as in network related domains. It serves for various purposes such as verifying the uniformity of the software functionality to the provided software specification, software performance check and troubleshooting the issues. Application log files or the logs generated by other monitoring tools are subjected to analysis for extracting information that can be vital in an investigation. These tasks demand competency to a great deal and are labor intensive when performed manually. There is no technique available to record expert knowledge and this stands as an obstacle to automate the analysis tasks. Hence there is a need for correlating information extracted from different locations in the same log file or multiple log files further adds to this complexity. This paper describes a practical approach for analysis of the logs using supervised learning to predict and to recommend steps for troubleshooting the issue. The overall solution proposed here is to automate the analysis of logs and provide recommendations for faster response to reported problems. The log analyser self-learns the new issue and provides necessary recommendations.

Keywords: - Log Analysis, Supervised learning, Rule engine, Data mining.

I. INTRODUCTION

Currently, the log analysis process has been more of manual task. The major pitfalls in the manual analysis process are that, it requires expertise, it is labour intensive, error prone and the advantage of recurrence is not used. A log contains error messages, application specific data and operational uses. In most applications the logs have different structures and different standards followed; the analysis of these logs requires tracing and finding any relevant pattern based on the reported problem in the log file. Especially for the support team the analysis of these logs becomes difficult and providing first level recommendation or troubleshooting becomes even more difficult. Apparently, the support team would need assistance from development team which incurs cost. The traditional approaches of manually analysing activities in the logs are time consuming and error prone. There is no tool that can continuously analyse and capture repetitive error occurring in the system. And there is no tool that provides recommendations for the engineer before he actually works on the issues. This will considerably increase the time invested for the analysis and troubleshooting process.

Here the challenges of analysing the existing log files using probabilistic analysis to provide aggregated information from the log files and issued alerts and recommendations based on the data present in the logs. The solution is designed with flexibility for future extension of any new generation or system log analysis with scalability in mind to process large log files.

II. RELATED WORKS

Work has been done in type casting the patterns of the logs from single or multiple files. Frequent pattern and association rule mining for semi-structured text data has been explored by many researchers [3]. Classification and clustering techniques for mining frequent patterns have been used for troubleshooting systems and software maintenance [5]. Most of these approaches apply supervised learning based on labelled training data, which is effective when there are a limited number of message formats that can be labelled.

Efforts have been made to provide a unified infrastructure for building expert systems that automate each step involved in log file analysis which formulates means for recording the knowledge of individual experts which can be later aggregated to build an aggregated knowledgebase in the form of mind maps [6].

All of the above approaches apply value-based clustering and a frame work to capture the knowledge. We use supervised learning, decision tree algorithm to classify the logs and to generate rules from the existing training data. Rule engine provides runtime to execute these rules.

III. METHODOLOGY

In this section, we discuss the approach for handling unstructured data in the log files, correlating the logs from multiple sources extract relevant patterns and match the patterns with the previous knowledge.

A. Algorithms

Classifier Algorithm: Decision tree is constructed from a fixed set of examples. The resulting tree is used to classify future samples. The example has several attributes and belongs to a class. The leaf nodes of the decision tree contain the class name whereas a non-leaf node is a decision node. The decision node is an attribute test with each branch being a possible value of the attribute. ID3 uses information gain to help it decide which attribute goes into a decision node. The advantage of learning a decision tree is that a program, rather than a knowledge engineer, elicits knowledge from an expert.

ReteOO: The Rete algorithm provides the basis for a more efficient implementation rule engine. A Rete-based expert system builds a network of nodes, where each node

except the root corresponds to a pattern occurring in the condition part of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern. This structure is essentially a generalized tire. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

B. Architecture overview

Figure 1 shows the architectural elements of our solution. The design is based on four major components. (1) The interpretation of the information of interest from both binary and text with an arbitrary structure and format. (2) Analytics Engine which calculates the uncertainty of each of the attributes in the knowledge base, using entropy, to form a decision tree and derives rules for information extraction from the formatted log. (3) Rule Engine provides runtime environment for validating the rules generated by the Analytics Engine against the formatted data. (4) Knowledge Base is the labelled training data.

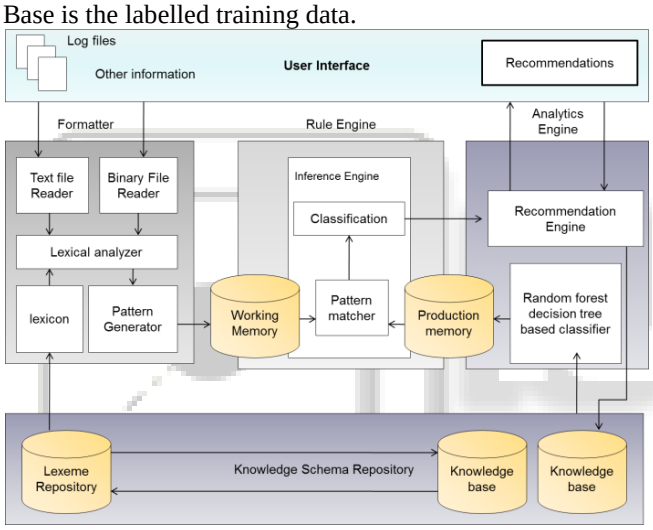

Fig. 1: Architecture elements

The first architecture element basically shows the data formatter engine which will process the unstructured or semi structured logs from the support dump to a universal format. The logs from various sources and platform have different file formats and also there is no standard followed in logging, which make it difficult in analyzing the logs. Formatter consists of lexical analyzer which searches for specific patterns available in the knowledge base and produces a structured data. The each attribute in the knowledge representation matrix is assigned with specific value from the knowledge base; the attributes are parameters like time stamp, source and destination IP for a network related log and source component, session id and other related information for application log. Regular expression is used to match the pattern.

The second architecture element shows the analytics engine which uses supervised learning to classify the log information using decision tree algorithm, sufficient set of training data is provided in the initial state. When the system is in training phase the engineer manually goes through the log file and creates an analogues structured data along with the label or classification, representing the issue and corresponding recommendation for the issue. Every time the log is analysed manually it is captured and represented in the knowledge base in the form of a knowledge representation matrix. Once sufficient data is available in the knowledge base, the label, representing the type of issue and corresponding recommendation for resolution, is classified using the decision tree algorithm. The best predictor forms the root node of the tree. The position of the nodes in the tree structure is decided by calculation the information gain of each of the attributes in the knowledge base; one with the highest information gain will be the root node. The rest of the nodes are computed by recursively calling decision tree. Information gain is obtained using the entropy of the attribute.

Entropy H(S) is a measure of the amount of uncertainty in the (data) set S

$$H(S) = -\sum_{x \in X} P(x) \log_2 P(x)$$

Where,

S - The current (data) set for which entropy is being calculated

X - Set of classes in S

P(x) - The proportion of the number of elements in class x to the number of elements in set S

Information gain IG (A) is the measure of the difference in entropy from before to after the set S is split on an attribute A.

$$IG(A) = H(S) - \sum_{t \in T} P(t)H(t)$$

H(S) - Entropy of set S

T - The subsets created from splitting set S by attribute A such that $S = \cup_{t \in T} t$

P(t) - The proportion of the number of elements in t to the number of elements in set S

H(t) - Entropy of subset t

a_1	a_2	a_3	..	a_n	C
v_{11}	v_{21}	..	..	v_{ij}	c_1
v_{12}	v_{22}	..	..		
..	..			v_{ij}	c_m

Where

$a_1, a_2, a_3, \ldots a_n$ ϵ A Attributes

$v_{11}, v_{12}, v_{13} \ldots v_{ij}$ ϵ V Values

$c_1, c_2, c_3 \ldots C_n$ ϵ C Classification

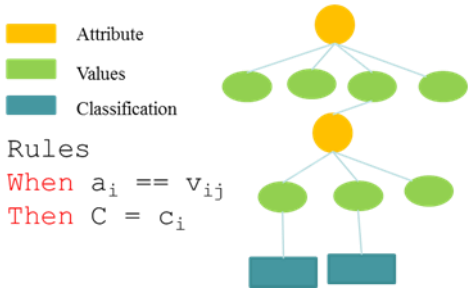

Fig. 2: Training data and decision tree

The knowledge is represented in the form of rules that are validated against the test data. The third architecture element is the rule engine, provides a runtime environment for firing the rules. The rules are derived by decision tree. The rules are both human as well as machine readable. Knowledge representation schema is the labelled data with

the structured entry for log file. The values and attributes serve as the tokens and lexeme in the lexical analyser.

C. Experimental setup

We prepared the prototype with java programming language along with JBoss Drools rule engine. We selected java because it had drools rule engine in java and was easy to write with many support functions. For rule engines, we selected Drools rule engine because it is one of best free rule engines with high performance. We needed to examine the algorithm. Then we decided to select some type of log to process, which reduce implementation time. The select logs to process were only web server log, DHCP log and radius log. However, the algorithm could support many type of computer logs with filtering support options.

The logs from multiple sources are correlated to and aggregated into generic format with all the relevant data from the previous knowledge. The below figure 3 shows the correlating the data from the multiple sources.

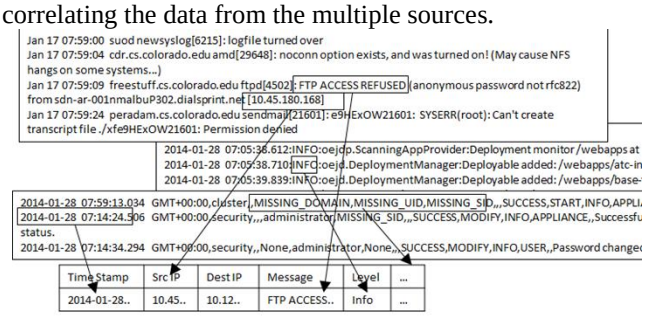

Fig. 3: Correlating and aggregating the logs.

The log from multiple sources where correlated to form universal format which was used to validate against the available knowledge. The below table shows the training data from the multiple log sources and correlated to a common format along with the label, representing which class it belongs. The decision tree algorithm is applied to classify the training data and the resulting classification tree represents the knowledge derived. The below figure shows the training data collected from correlating web server log, DHCP log and radius logs. And these are classified into two class.

Level	IP Address	Atentication	Upload Error	Classification
Error	10.63.140.37	Failed	ermission denied	WA_1000
Warn	10.63.140.37	Failed	ermission denied	WA_1000
Error	10.63.140.37	Failed	Memory full	WA_1001
Error	10.45.14.25	Failed	TP Access Refuse	WA_1001
Error	10.75.24.63	SUCCESS	TP Access Refuse	WA_1001
Warn	10.75.24.63	SUCCESS	TP Access Refuse	WA_1000
Warn	10.75.24.63	SUCCESS	Memory full	WA_1001
Error	10.45.14.25	Failed	ermission denied	WA_1000
Error	10.75.24.63	SUCCESS	ermission denied	WA_1001
Error	10.45.14.25	SUCCESS	TP Access Refuse	WA_1001
Warn	10.45.14.25	SUCCESS	ermission denied	WA_1001
Warn	10.45.14.25	Failed	Memory full	WA_1001
Error	10.63.140.37	SUCCESS	Memory full	WA_1001
Warn	10.45.14.25	Failed	TP Access Refuse	WA_1000

Fig. 4: Sample training data in universal format

The derived knowledge from the decision tree is used to generate the rules the below figure shows rules generated for Drools rule engine that are validated against the new logs.

```
rule "classification WA_1000"
    when
        LogEntry(
            isError("permission denied") &&
            isAtentication("Failed") ||
            isError("FTP Access Refused") &&
            isLevel("Error")
        );
    then
        setClass("WA_1000");
end

rule "classification WA_1001"
    when
        LogEntry( isUploadError("permission denied") &&
        isAtentication("SUCCESS") ||
        isUploadError("Memory_full") ||
        isUploadError("FTP Access Refused") &&
        isLevel("Warn"));
    then
        setClass("WA_1000");
end
```

Fig. 5: Sample training data in universal format

The figure below shows self-explanatory sequence diagram for the log analyser.

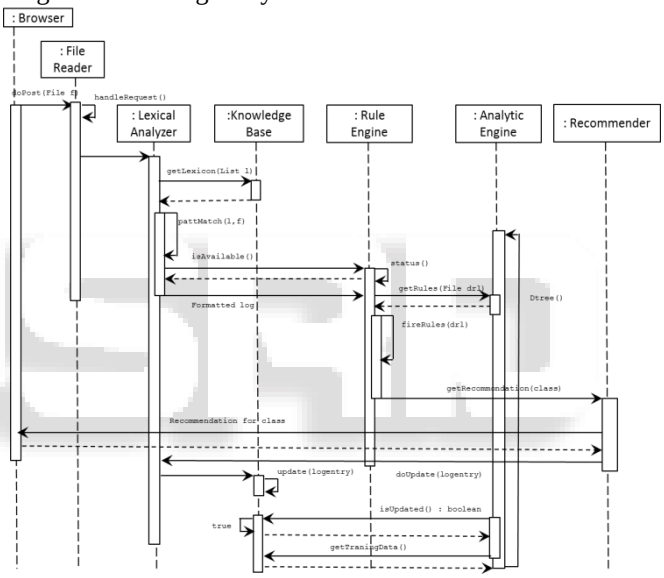

Fig. 6: Sequence diagram

IV. Result and Future Work

The proposal of adaptive log analyser based on rule engines is successfully completed and validated for various log file formats. The decision tree perfectly classifies the training data which is used by the rule engine. The recommendations given by the system is observed to give Maximum percentage of correct results.

The system was built for Tomcat container and could handle logs of smaller size. The current setup thought to be upgraded and deployed on hadoop cluster which can be used to leverage the map reduction technique as a part of future work. This can make the log analyser highly scalable. Also, Multiple logs can be analysed in parallel.

References

[1] Nathaphon Kiatwonghong, Songrit Maneewongvatana. 2010 2nd International Conference on Education Technology and Compute. Intelli-Log : A Real-time Log Analyzer.

[2] CAPRI: type-CAsted Pattern and Rule mIner. 2013. At: http://research.cs.queensu.ca/home/farhana/capri.html.

[3] Fu, Q., Lou, J.G., Wang, Y., and Li, J., 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In proc. of the IEEE ICDM, pp. 149 – 158, Miami, FL.
[4] Hellerstein, J., Ma, S., and Perng, C., 2002. Discovering Actionable Patterns in Event Data. IBM System Journal, vol. 41(3).
[5] Jiang, Z., Hassan, A., Hamann, G, and Flora, P., 2008. An Automated Approach for Abstracting Execution Logs to Execution Events. Journal of Software Maintenance and Evolution: Research and Practice, vol. 20, pp. 249–267.
[6] Dileepa Jayathilake. 2011 International Conference on Software and Computer Applications IPCSIT vol.9, Singapore.

