<LOG DATA ANALYSIS>

A PREPRINT

Domenico Girodano
Department of <>
<To Be Filled>
Santa Narimana, Levand
domenico.giordano@cern.ch

Ravi Charan
Department of Information Technology
Indian Institute of Information Technology
Allahabad, India
ravicharan.vsp@gmail.com

July 18, 2019

ABSTRACT

Massive amount of data is generated by the Openstack cloud services in the format of service logs. Besides timestamps and log level fields, these logs contain additional information useful for pattern analysis. Unfortunately this information is generally exposed in semi-structured text format, not allowing direct analysis without additional munging of the data. Traditional approaches to extract information from those fields are rule-based, mainly applying regular expressions upon knowledge of the text structure. These approaches require a pre-knowledge of all text patterns and are not scalable with the services growth. We propose to build a data analytic approach for log data, leveraging the technologies adopted for Web [1] and genomics [2] data mining (such as minhashing [3] and locality-sensitive hashing [4])

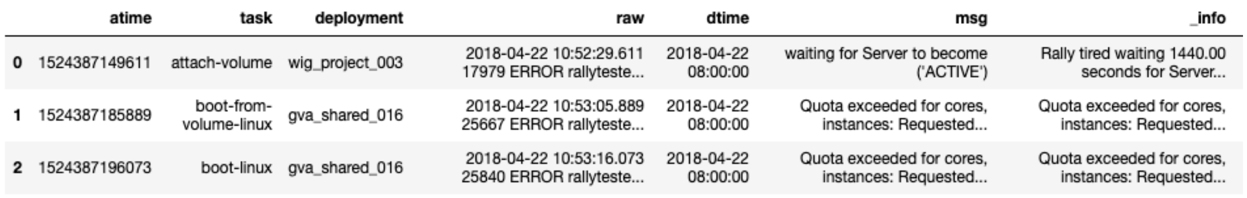
1 Introduction

Analysing logs to identify patterns and problems has been attempted ever since logs have been generated. At its most general, to log is ”to put in- formation into a written record” and is a process as old as writing systems themselves. Previously, manual investigation was used whereby administrators would read the logs stored on each computer, perhaps aided by grepping for keywords. As the scale of services has increased in the age of Big Data, this approach is no longer feasible.

Attempts have been made in the last 20 years to automate this process. One elementary method that is investigated regularly and is a classic ex- ample of expert systems that emulate the human expert decision making process is the use of regular expressions (regex), nested if-then rules used to select key features. However, this requires prior knowledge of the dataset, is proportional in difficulty to the number of distinct message types present and require continuous updating if the nature of log files changes.

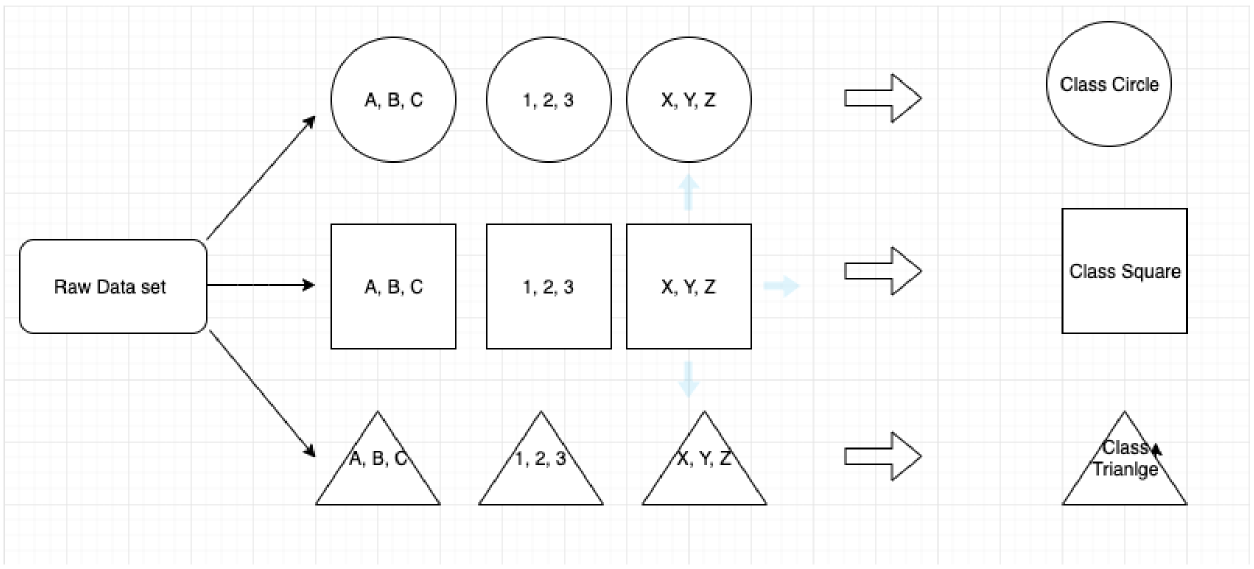
Alternatievly, supervised and unsupervised machine learning algorithms have been investigated. In supervised learning training datasets where anomalous and normal data are labelled by hand are used to train an algorithm. Decision trees[5][3] have been used in this regime for classification of logs files when the error categories are already known. Since logs are generally labelled with a time stamp, a different school if techniques have utilised techniques adapted to time series data. . A Finite State Automata was trained assuming that each log key can be mapped to a state transition.

In this way, the chronology of interrelated messages is captured, and anomalies in transition timings and loop executions are used to identify system errors. This method is relatively successful but requires that edit distances between all logs is calculated which is infeasible for large datasets. We majorly deal with the log data which contains the ”ERROR” messages and these messages are analyzed and then clustered into classes and example of the log error messages produced could be seen below.



	atime	task	deployment	raw	dtime	msg	_info
0	1524387149611	attach-volume	wig_project_003	2018-04-22 10:52:29.611 17979 ERROR rallyteste...	2018-04-22 08:00:00	waiting for Server to become ('ACTIVE')	Rally tired waiting 1440.00 seconds for Server...
1	1524387185889	boot-from-volume-linux	gva_shared_016	2018-04-22 10:53:05.889 25667 ERROR rallyteste...	2018-04-22 08:00:00	Quota exceeded for cores, instances: Requested...	Quota exceeded for cores, instances: Requested...
2	1524387196073	boot-linux	gva_shared_016	2018-04-22 10:53:16.073 25840 ERROR rallyteste...	2018-04-22 08:00:00	Quota exceeded for cores, instances: Requested...	Quota exceeded for cores, instances: Requested...

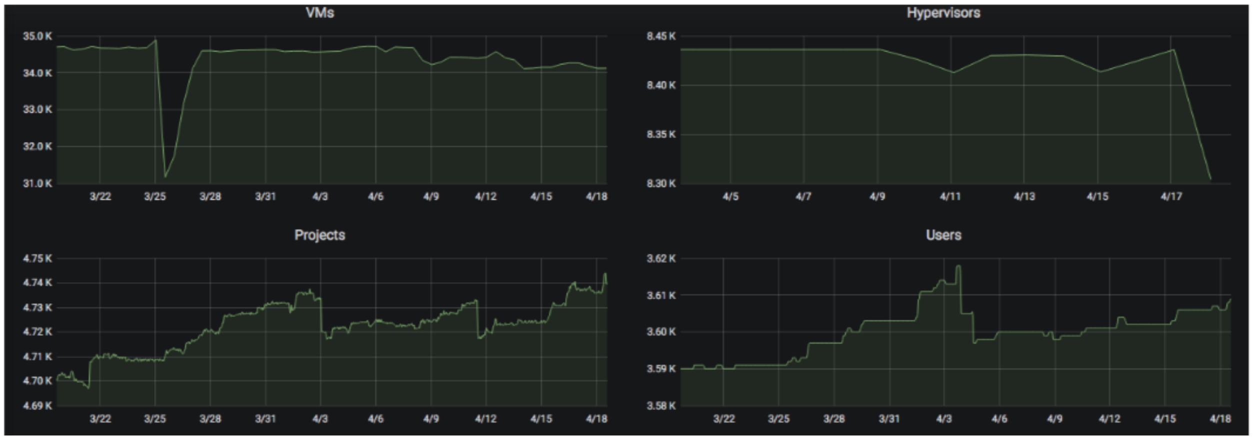
The basic idea is to analyse the data present and cluster the messages in to different classes. One can see from the error logs that a few features of all the messages are unique (eg. times- tamps, ErrorID.. etc). These unique features are given lesser priority in the clustering of the data hence corresponding weights are assigned to these features. The distance between the messages is compared by ignoring these unique features. The trivial logic is to put the messages whose distance is less than a predefined threshold are put into the same class. A dummy illustration of the algorithm is shown below.



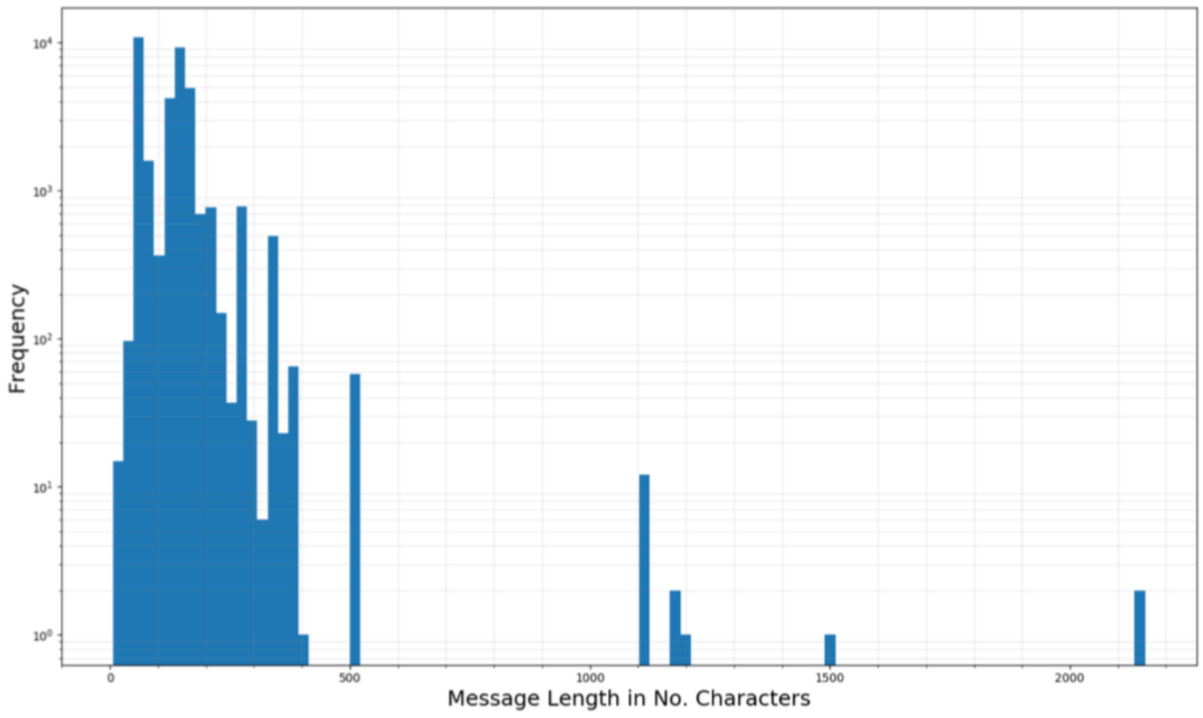
The figure shows the demonstration of the clustering and the classification procedure. Each one of the 3 shapes represents as class and the variables inside each shape represent the unique features in the dataset. The data is clustered based on how similar the shapes are ignoring the unique features of the message. Once the messages are clustered, the principal features are extracted from each class and marked as a feature vector of that class. Now, every new entry in the data point is compared with these principal feature vectors to classify the new entry into one of the classes.

2 Dataset

In the operation of a data-center, being able to categorise errors automatically in order to take curative action and/or for monitoring purposes would be highly valued. One technology that forms the foundation through which a significant portion of CERN's computing resources are distributed is Open- Stack. In essence, rather than assigning physical machines to users, much larger facilities are purchased and Virtual Machines (VM's) are generated with fractions of available resources and assigned to users. Tens of thousands of VM's are operating at any one time, as can be seen in in the figure below, and in the process vast quantities of logs are generated in the form of short descriptions as strings composed of words. The logs are categorized at the service level as either monitoring messages that are logging normally occurring events (such as VM generation progress reports) or error messages, representing undesired issues.



In the creative destruction of the daily churn of machines, vast quantities of logs are generated, and each varying according to the service. These logs are a valuable resource since they were coded by the original creators of the services and are the main source of error reporting in OpenStack. Given that on average 100k logs are produced every second generating 3TB's of data per day, when presented with a new message it can be time consuming to sort through the vast body of historical logs manually to find similar messages.



3 Methodologies

The first step is to extract message "keys" which correspond to messages generated by the same lines of code but with different message vari- ables, implemented using regex. They proceed to utilise the similarity in clustering raw log keys by using the edit distance as their metric, which is defined as the number of insertion, deletion or re- placement operations required to convert one mes- sage into another. This distance does not account for the position of the words in the logs, and so they use the weighted edit distance which uses sig- moid function to place greater emphasis on words appearing earlier in the message than later based on the idea/observation that programmers tend to place important information in this way. The edit distances to all other logs is calculated, and all members whose distance is less than a threshold is connected via a link.

In this section, we define how we approach the problem mathematically, starting with our choice of similarity measure, the Jaccard Similarity, J: in order to identify similar items, we must first identify how to define and quantify similarity. We progress to describe shingling, a nave method to fully iterate over the parameter space in order to compute J, and learn how it is infeasible to apply to large datasets, such as when hundreds of thousands of logs are generated every minute, as can be the case in distributed systems.

3.1 MinHash

In order to identify similar items, a similarity measure is necessary. A common and intuitive means is that of the Jaccard Similarity. This treats two strings as sets of words, and defines the size of their intersection relative to their union as the defining metric of similarity. We therefore see that not only can the relation between two logs be evaluated,but the distances between different logs represent hierarchies of similarity, and these can be mined to reveal underlying clusters. Jaccard similarity J between two given set of words A and B is given by

$$J = |A \cap B| / |A \cup B| \tag{1}$$

Other alternative methods include measuring distance as the number of edits required to transform one message in another. he signatures we desire to construct for sets are composed of the results of a large num- ber of calculations, say several hundred, each of which is a minhash. To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows. The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1. The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets. **[1]**

3.1.1 Locality Sensistive Hashing

MinHash is an effective means for reducing the dimensionality of variable length alphanumerical data to fixed length numerical output, with which Jac- card similarity can be probabilistically calculated via random sampling. However, each log hash must still be compared to every other in order to identify relationships, resulting in complexity O(n**2). There are a range of search methods for identifying nearest neighbours (examples, references) based on space partitioning but it was shown that they all degrade to linear search for sufficiently high dimensions. Locality Sensitive Hashing is used to reduce the search space by using hash collisions as a proxy for similarity. **[Add reference for LSH here]**

3.1.2 Shingling

TO be Filled

4 Comparative study

We implemented multiple approaches to implement the clustering of the log data and are tested for better performances in terms of accuracy and efficiency. The methods implemented include Minhash LSH (tested with shingles of variable sizes), and a Graph based approach that treats all the messages as nodes and the weight of the edges as the distance between the messages. K-Means was rules out from the comparative study because of the constraint with respect to the number of classes that has to be specified into the algorithm. Since the Openstack data set is vast, we do not have the information about the number of classes of the error messages.

4.1 Weighted MinHash

4.2 Shingling

4.3 Graph Based Clustering methods

References

[1] Andrei Z. Broder, Identifying and Filtering Near-Duplicate. Documents AltaVista Company, San Mateo, CA 94402, USA

[2] Name 1 and name2. Title. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.