Unsupervised Noise Detection in Unstructured data for Automatic Parsing

Shubham Jain*, Amy de Buitléir†, and Enda Fallon*
*Software Research Institute, Athlone Institute of Technology, Athlone, Ireland
sjain@ait.ie efallon@ait.ie
†Network Management Lab, Ericsson, Athlone, Ireland
amy.de.buitleir@ericsson.com

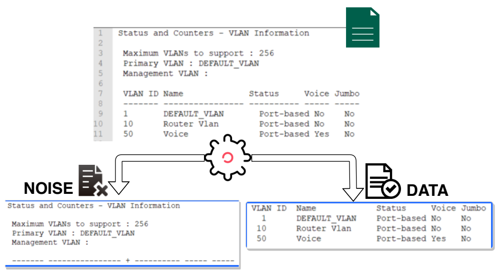
***Abstract*—The telecommunications industry makes extensive use of data extracted from logs, alarms, traces, diagnostics, and other monitoring devices. Analyzing the generated data requires that the data be parsed, re-structured, and re-formatted. Developing custom parsers for each input format is labor-intensive and requires domain knowledge. In this paper, we describe a novel unsupervised text processing pipeline to automatically detect and label relevant data and eliminate noise using Levenshtein similarity and Agglomerative clustering. We experiment with different similarity and clustering algorithms on a selection of common data formats to verify the accuracy of the proposed technique. The results suggest that the proposed methodology has higher accuracy.**

***Index Terms*—Unsupervised Data Mining, Information Extraction, Clustering, Similarity**

I. Introduction

Telecommunications networks contain a wide variety of types of equipment, each of which may generate large volumes of diagnostic information, including system logs, alarms, traces, diagnostics, and other monitoring devices. Much of the data is in free-form text format, and there is a business need to automatically identify useful information and convert it to a format that can be used to monitor and maintain the network. Network monitoring requires data exploration and machine learning applications such as anomaly detection [1], cognitive management [2], incident management, autonomous problem validation [3], root cause analysis and many more. Such applications can identify and diagnose network issues and generate insights that can help optimize the network and make network management more efficient. With the growing complexities of the network due to 5G and virtualization, it is important to find an efficient solution for processing this free-form data [4].

Currently, handling free form textual data has two approaches, 1) manual handling by domain experts and 2) natural language processing with supervised learning. Manual handling requires the development and maintenance of a large number of custom parsers and thus is labor-intensive. The natural language approach has been widely researched and implemented in recent years [5]. However, this method requires extremely large labeled (annotated) training sets, which are labor-intensive to create. A simple illustration is shown in Fig. 1 where a text document is analyzed, and the resulting output has noise and data separated. As seen in the figure, noise are the input lines 1-6 and 8 are extraneous; they are not useful for exploratory data analysis. In contrast, lines 7 and 9-11 contain information that can be parsed and analyzed.



```
1   Status and Counters - VLAN Information
2
3    Maximum VLANs to support : 256
4    Primary VLAN : DEFAULT_VLAN
5    Management VLAN :
6
7    VLAN ID Name             Status     Voice Jumbo
8    ------- ---------------- ---------- ----- -----
9    1       DEFAULT_VLAN       Port-based No    No
10   10      Router Vlan        Port-based No    No
11   50      Voice              Port-based Yes   No
```

```
Status and Counters - VLAN Information

 Maximum VLANs to support : 256
 Primary VLAN : DEFAULT_VLAN
 Management VLAN :

 ------- ---------------- + ---------- ----- -----
```

```
VLAN ID   Name             Status      Voice  Jumbo
  1       DEFAULT_VLAN     Port-based No     No
 10       Router Vlan      Port-based No     No
 50       Voice            Port-based Yes    No
```

Fig. 1: An Illustrative Example of Noise detection

In this paper, we introduce an unsupervised text analysis pipeline that identifies extraneous information and segregates it from relevant data and parses it into a structured format. The technique is domain-independent and does not require a training set. It is resilient to changes in the structure and formatting of data. The process is automatic and requires no input from a domain expert. A similarity matrix is created and used to cluster the lines of text. Each cluster is identified as containing either noise or relevant information for parsing. We compare combinations of various similarity and clustering algorithms and present each combination's accuracy in this paper's later section. Towards the end, we provide the most accurate and optimal pipeline with similarity and clustering algorithms for unsupervised noise detection and parsing.

This paper is organized as follows: §II describes an overview of existing approaches for data parsing. We present our methodology in §III. We discuss our evaluation method and compare results of different combinations in §IV and conclude in §V.

II. Related Work

In this section, we discuss the different approaches for data extraction from free-form data. These methods can be classified into heuristics-based and machine learning-based techniques.

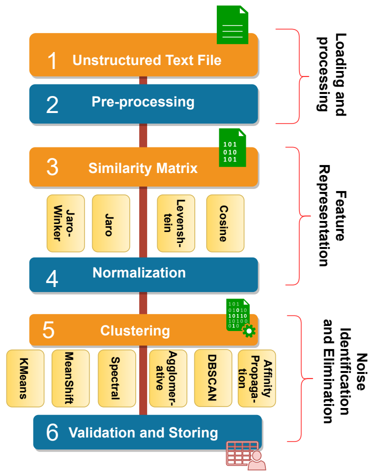


Fig. 2: Methodology: Unstructured Data Analysis and Parsing Pipeline

Heuristics-based parsing was presented by the authors of Iterative Partitioning Log Mining (IPLOM) [6]. They focused on template-construction by using a three-step hierarchical process; partitioning, word distribution and splitting the messages into the constant and variable part. Another heuristics-based approach was implemented by the authors of Log key Analysis (LKE) [7] where they clustered messages based on pre-defined weights and rules. Both these approaches worked well for similar structures in a data file. These approaches are not generic enough to apply tor different structures across a large scale system and manual updates are required when the structure of data changes.

The authors of [8] evaluated various rule-based parsers such as SLCT [9], IPLOM, LogSig [10] on 16 datasets and concluded that log analysis techniques are not accurate and further claimed that even slight error in pre-processing can cause a huge performance degradation on applications such as anomaly detection [11] [12] [1], cognitive management [2], autonomous problem validation [13] [3], incident management or root cause analysis.

An unsupervised approach using Long-short-Term-Memory Neural Network was introduced by the authors of LogRobust [14]. They converted each log line into fixed dimension semantic vector. The sequence of semantic vectors were used to process noise during collection and pre-processing of data. However, the approaches only works well with linguistic messages and falls short on identifying and filtering components such as timestamp, separators and other extraneous lines in an unevenly distributed file format. Furthermore, the approaches are sensitive towards change in the structure of the file and require manual cleaning of a file before processing it for automatic parsing. We presented a detailed review of the applications and technologies for unstructured data analysis in [15].

III. Methodology

In this section, we present the implementation of our noise detection pipeline that is domain-independent, segregates extraneous information, and parses the required format without the need for labeled training data.

As illustrated in Figure 2, the proposed noise detection pipeline consists of three main phases. Phase I focuses on loading our data and removing redundant spaces and special characters in phase II, where each line is compared with all the other lines in the file, to generate a similarity matrix. This similarity matrix is then normalized and provided as an input to the clustering algorithm in phase III. The clustering algorithm identifies two different clusters, and based on heuristic rules, the cluster containing noise is identified. The detailed description of each phase is provided in the next sections.

A. Feature Representation

We preprocess the file and remove consecutive special characters and white-spaces using regular expressions. After processing the file, we use similarity algorithms to compare each line with other lines inside the file. We experimented with several edit-distance-based similarity algorithms such as Levenshtein distance [16] to calculates the number of operations such as insertions, updations or deletions required to transform string x into string y where x and y are lines of length $|x|$ and $|y|$ in our text file. resulting distance lies on the closed interval [0,1], where 0 indicates no similarity and 1 indicates complete equality between lines. The formula for Levenshtein distance is given in Equation 1

$$lev_{x,y} = \begin{cases} max(i,j) & if\ min(i,j) = 0 \\ min \begin{cases} lev_{x,y}(i-1,j) = 1 \\ lev_{x,y}(i,j-1) = 1 \\ lev_{x,y}(i-1,j-1) = 1 \end{cases} & otherwise \end{cases} \tag{1}$$

Jaro distance is another similarity algorithm; it computes the number of common characters in two strings and identifies patterns in character deviation [17]. The Jaro similarity of two strings x and y is given in Equation 2

$$sim_j = \begin{cases} 0 & if\ c = 0 \\ \frac{1}{3}\left(\frac{c}{|x|} + \frac{c}{|y|} + \frac{c-t}{c}\right) \end{cases} \tag{2}$$

where c is number of matching characters and t is the number of transpositions from x and y.

Jaro-Winkler [18] is another edit-distance-based similarity algorithm that uses a fixed prefix scale p yielding much better scores to strings that match from the beginning index. The Jaro-Winkler similarity of two strings x and y is given in Equation 3

$$sim_w = sim_j + lp(1 - sim_j) \tag{3}$$

where l is the length of prefix and p is the prefix in the beginning of a line. We set a default value 0.1 to p as a value greater than 0.25 can make similarity larger than 1.

Furthermore, we convert our strings x and y to vectors x^v and y^v and implement Cosine Similarity that computes the cosine angle between two nonzero vectors. The cosine similarity ranges from [-1,1] where -1 indicates no similarity and 1 indicates total similarity and is given in Equation 4.

$$cos(\theta) = \frac{x^v \cdot y^v}{||x^v||\ ||y^v||} \tag{4}$$

Algorithm 1 Similarity Matrix

```
1:  procedure CALCULATESIMILARITY
2:      Algorithm ← [Levenshtein, Jaro, Rosetta Jaro, Cosine]
3:      df ← Read File
4:      ScoreList ← []
5:      for df.iterrows() do
6:          for $df_i$.iterrows() do
7:              $x_i$ ← df[data].Read Line 1
8:              $y_{i+1}$ ← df[data].Read Line 2
9:              score ← callAlgorithm($x_i, y_i$)
10:             ScoreList ← append(score)
11:             i ← i + 1
12:         end for
13:         $df_i$ ← append(ScoreList)
14:     end for
15: end procedure
```

A feature representation matrix is generated by using the similarity algorithms describes above. Our algorithm is illustrated in Algorithm 1. We start reading lines from the input file and pass them as parameters to our desired similarity algorithm that returns a similarity score. We then compare each line's score and create a list representing a similarity score of $line_i$. Thus, if there are ten lines in the input text file, then this will generate ten different lists, merged to form a matrix of comparison between each line. After the matrix was generated, there was a need to normalize the scores and bring them to a range of [-1,1]. Some algorithms, such as Jaro, Jaro-Winkler, and Cosine, generated a matrix with close minimum and maximum values, but Levenshtein generated outputs with a higher difference in minimum and maximum values. Thus, we applied *SimHash* [19] to our similarity matrix as it performs dimensionality reduction by putting similar vectors closer in a low-dimensional vector space. Our experiments and results are discussed in §IV. The output of the most efficient algorithm is identified to facilitate downstream noise detection algorithms.

B. Noise detection and Elimination

Since our main objective is to identify noise from a collection of lines generated from various nodes, we perform clustering to segregate relevant data and noise. We implemented six clustering algorithms from the sklearn.cluster module of scikit-learn [20] to identify the best combination of clustering and feature representation in the pipeline. Table I describes a summary of clustering algorithms and provides details on the clustering algorithm's metric. The last column provides information on the clustering algorithms that identify the number of clusters in our data without user input.

Algorithm	Metric	Automatic Clustering
K-Means	Distance-based	N
Mean Shift	Distance-based	Y
Spectral	Graph-based	N
Agglomerative	Pair-wise distance	N
DBSCAN	Nearest-Point Distance	Y
Affinity propagatoion	Graph-based	Y

TABLE I: Summary of Clustering Algorithms

Clustering Algorithm	Accuracy%	Accuracy(SimHash)%
K-Means	43	44
Mean Shift	56	51
Spectral	71	88
Agglomerative	90	62
DBSCAN	84	68
Affinity propagation	76	54

TABLE II: Accuracy: Jaro Similarity with Clustering Algorithms

Our method focuses on forming two clusters, one with noise and other with relevant data that can be parsed and analyzed. We also experimented with three clusters to detect components such as noise, data, and header.

Finally, after we cluster our data, we generate labels for each line where cluster one is labeled as **0**, and cluster two is labeled as **1**. We assign these labels to each line and generate two files segregating both the cluster labels. We run both the files through a set of rules that identifies the cluster that contains noise. Our rules focus on identifying lines with consecutive white-space at an index followed by a comparison of line sizes in each cluster. The data required for analysis in table form and contains components such as header, rows, columns, and cells. In an individual sample, the data rows have a similar number of columns, and comparing the mean of the size of each row with the size of each row had a higher difference in the noisy cluster. The algorithm for identification of the noisy cluster is presented in Algorithm 2.

IV. Evaluation and Discussion

In this section, we present the results of the methodology discussed in the earlier section. We present the most accurate algorithm for the generation of the similarity matrix and clustering. We also discuss the dataset and evaluation metrics for analyzing the results of our unsupervised noise detection algorithm. We present the accuracy for each combination of pipeline and present it in the results subsection.

A. Dataset and Evaluation

We use a collection of an anonymized dataset from a telecommunications vendor and various open-source samples

Algorithm 2 Noise Detection after clustering

```
1: procedure CALCULATESIZE(inputdf)
2:     df ← inputdf
3:     SizeList ← []
4:     for df.iterrows() do
5:         line ← df[data].Read
6:         lineSize ← len(len)
7:         SizeList ← append(lineSize)
8:         df ← append(SizeList)
9:     end for
10: end procedure
11: procedure CALCULATEDIFF(inputdf)
12:     df ← inputdf
13:     DiffList ← []
14:     mean ← [SizeList].mean()
15:     for df.iterrows() do
16:         Size ← df[SizeList].Read
17:         difference ← Size - Mean
18:         DiffList ← append(difference)
19:         df ← append(DiffList)
20:     end for
21: end procedure
22: procedure IDENTIFYNOISECLUSTER(inputdf0,inputdf1)
23:     df0 ← inputdf0
24:     df1 ← inputdf1
25:     meanDiff0 ← df0[diffList].mean()
26:     meanDiff1 ← df1[diffList].mean()
27:     if $meanDiff0 > meanDiff1$ then
28:         $noiseCluster \leftarrow 0$
29:     else
30:         $noiseCluster \leftarrow 1$
31:     end if
32: end procedure
```

Clustering Algorithm	Accuracy%	Accuracy(SimHash)%
K-Means	77	74
Mean Shift	68	54
Spectral	51	54
Agglomerative	86	78
DBSCAN	91	73
Affinity propagation	44	30

TABLE III: Accuracy: Jaro-Winkler Similarity with Clustering Algorithms

Clustering Algorithm	Accuracy%	Accuracy(SimHash)%
K-Means	62	76
Mean Shift	45	42
Spectral	65	44
Agglomerative	**96**	84
DBSCAN	90	68
Affinity propagation	85	73

TABLE IV: Accuracy: Levenshtein with Clustering Algorithms

Clustering Algorithm	Accuracy%	Accuracy(SimHash)%
K-Means	71	65
Mean Shift	54	42
Spectral	67	44
Agglomerative	82	79
DBSCAN	84	81
Affinity propagation	77	77

TABLE V: Accuracy: Cosine Similarity with Clustering Algorithms

from ntc-templates[1] that provides free-form, unstructured text from a variety of network vendors to evaluate each pipeline of algorithms. We labelled each line with noise and data labels to evaluate the accuracy of our unsupervised approach.

Our algorithm identifies lines that resemble a table-like structure in a single unstructured text-heavy file. Our pipeline processed an individual sample file and categorized it into noise and relevant data. Our evaluation method consists of manually labeling each input and determining the accuracy of the processed file by comparing it with the manual labels. We then take a mean of the accuracy of the 50 samples and calculate F1-score [21] for each pipeline to identify the pipeline's actual accuracy.

B. Results

Tables II through V report the accuracy of each pipeline in this section. Each table presents two accuracy metrics, one with the unmodified similarity matrix, and one with SimHash applied to the similarity matrix. Table II shows the accuracy of the pipeline where the matrix is generated using the Jaro similarity algorithm with the combination of different clustering algorithms and normalized data. The accuracy of clustering algorithms with Jaro-Winkler is shown in Table III. Table IV and Table V shows the accuracy of the pipeline with Levenshtein and Cosine similarity algorithms with clustering algorithms, respectively.

The results suggest that the pipeline that detects noise with the highest accuracy is the one where the feature representation matrix is generated using the Levenshtein distance algorithm, and the matrix is clustered using Agglomerative hierarchical clustering. The lower accuracy when SimHash was used suggests that dimensionality reduction removes the necessary features required for the classification of data.

V. CONCLUSIONS

This research describes a text processing pipeline to identify extraneous information and segregate it from relevant data. The proposed pipeline behaves as an intermediary step between data collection and data analysis. This approach categorizes lines without the need for a domain expert and does not require labeled training data. This research provides a significant improvement over traditional data cleaning methods by domain experts in terms of cost and resources.

[1]https://pypi.org/project/ntc-templates

Acknowledgments

This work is funded by Irish Research Council Enterprise Partnership Scheme Postgraduate Scholarship 2020 under Project EPSPG/2020/7.

References

[1] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, oct 2016.

[2] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, jan 2018.

[3] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 402–411.

[4] S. Satpathi, S. Deb, R. Srikant, and H. Yan, "Learning latent events from network message logs," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1728–1741, 2019.

[5] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "Deepdesrt: Deep learning for detection and structure recognition of tables in document images," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 1162–1167.

[6] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2011.

[7] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 ninth IEEE international conference on data mining*. IEEE, 2009, pp. 149–158.

[8] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 654–661.

[9] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. IEEE, 2003, pp. 119–126.

[10] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 785–794.

[11] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 117–132.

[12] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, dec 2009.

[13] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 267–277.

[14] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[15] S. Jain, A. de Buitléir, and E. Fallon, "A review of unstructured data analysis and parsing methods," in *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2020, pp. 164–169.

[16] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, Feb 1966.

[17] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989.

[18] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage." 1990.

[19] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002, pp. 380–388.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] T. Joachims, "A support vector method for multivariate performance measures," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 377–384.