Master of Science in Computer Science
January 2021

Automating Log Analysis

Sri Sai Manoj Kommineni
Akhila Dindi

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Sri Sai Manoj Kommineni
E-mail: srko18@student.bth.se

Akhila Dindi
E-mail: akdi18@student.bth.se

University advisor:
Dr. Hüseyin Kusetogullari
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background: With the advent of information age, there are very large number of services rising which run on several clusters of computers. Maintaining such large complex systems is a very difficult task. Developers use one tool which is common for almost all software systems, they are the console logs. To trouble shoot problems, developers refer to these logs to solve the issue. Identifying anomalies in the logs would lead us to the cause of the problem, thereby automating analysis of logs. This study focuses on anomaly detection in logs.

Objectives: The main goal of the thesis is to identify different algorithms for anomaly detection in logs, implement the algorithms and compare them by doing an experiment.

Methods: A literature review had been conducted for identifying the most suitable algorithms for anomaly detection in logs. An experiment was conducted to compare the algorithms identified in the literature review. The experiment was performed on a dataset of logs generated by Hadoop Data File System (HDFS) servers which consisted of more than 11 million lines of logs. The algorithms that have been compared are K-means, DBScan, Isolation Forest and Local Outlier Factor algoritms which are all unsupervised learning algorithms.

Results: The performance of all these algorithms have been compared using metrics precision, recall, accuracy, F1 score and run time. Though DBScan was the fastest, it resulted in poor recall, similarly Isolation Forest also resulted in poor recall. Local Outlier Factor was the fastest to predict. K-means had the highest precision and Local Outlier Factor had the highest recall, accuracy and F1 score.

Conclusion: After comparing the metrics of different algorithms, we conclude that Local Outlier Factor performed better than the other algorithms with respect to the most of the metrics measured.

Keywords: Anomaly detection, Log analysis, Unsupervised learning

Acknowledgments

We would like to show our sincere gratitude to our academic supervisor Dr. Hüseyin Kusetogullari for supervising and giving us useful feedback. We would also like to thank our company supervisors Yael Katzenellenbogen and Martha Dabrowska for advising us throughout our thesis. We would also like to extend our gratitude to our friends and family who supported and helped us directly and indirectly.

Contents

Abstract i

Acknowledgments iii

Contents v

List of Figures vii

List of Tables viii

List of Equations ix

1 Introduction 1
- 1.1 Aim . . . 2
- 1.2 Objectives . . . 3
- 1.3 Research questions . . . 3
- 1.4 Defining the scope of the thesis . . . 3
- 1.5 Outline . . . 3

2 Background 5
- 2.1 Anomaly Detection . . . 5
- 2.2 Machine Learning . . . 6
  - 2.2.1 Supervised Machine Learning . . . 6
  - 2.2.2 Unsupervised Machine Learning . . . 6
  - 2.2.3 Semi-Supervised Machine Learning . . . 7
  - 2.2.4 Reinforcement Machine Learning . . . 7
- 2.3 Word embeddings . . . 8
- 2.4 Algorithms . . . 8
  - 2.4.1 GloVe: . . . 8
  - 2.4.2 Cosine similarity . . . 8
  - 2.4.3 K-means-algorithm . . . 9
  - 2.4.4 DBSCAN . . . 10
  - 2.4.5 Isolation Forest . . . 11
  - 2.4.6 Local outlier factor . . . 12
  - 2.4.7 k-Nearest Neighbour (KNN) Algorithm . . . 13
- 2.5 Performance metrics . . . 14
  - 2.5.1 Accuracy . . . 14
  - 2.5.2 Precision . . . 15
  - 2.5.3 Recall . . . 15

2.5.4 F1 score . . . 15
2.5.5 Run time . . . 15

3 Related Work **17**

4 Method **23**
4.1 Literature Review . . . 23
4.1.1 Investigation of primary studies . . . 24
4.1.2 Criteria for the selection of research . . . 24
4.1.3 Assessment of quality . . . 25
4.1.4 Extraction of data . . . 25
4.2 Experiment . . . 25
4.2.1 Experiment Set-up / Tools used . . . 26
4.3 Data Collection . . . 27
4.3.1 Dataset . . . 27
4.3.2 Data Preprocessing . . . 27
4.3.3 Log Parsing . . . 28
4.3.4 Feature Extraction . . . 29
4.4 Algorithms . . . 31
4.4.1 K-means algorithm . . . 31
4.4.2 DBSCAN Algorithm . . . 32
4.4.3 Isolation Forest . . . 33
4.4.4 Local Outlier Factor . . . 34
4.5 Testing . . . 35

5 Results and Analysis **37**
5.1 K-means algorithm . . . 37
5.2 DBSCAN algorithm . . . 38
5.3 Isolation Forest algorithm . . . 38
5.4 Local Outlier Factor algorithm . . . 39
5.5 Comparing Algorithms . . . 40

6 Discussion **43**
6.1 Answering RQ1 . . . 43
6.2 Answering RQ2 . . . 43
6.3 Validity Threats Analysis . . . 44

7 Conclusions and Future Work **45**

Bibliography **47**

A Supplemental Information **55**

List of Figures

2.1 *DBSCAN*[84] 10
2.2 *IsolationForest*[31] 11
2.3 *Local Outlier Factor*[25] 13

4.1 *Working method of Anomaly detection* 27
4.2 *Before parsing the Console Logs* 28
4.3 *Parsed Logs* 29
4.4 *Features Extracted* 30
4.5 *Distribution of data over distance, x-axis: frequency of datapoints, y-axis: distance from the centre of cluster* 32
4.6 *Distribution of data over distance, x-axis: data point index value ranging from 0 to 100,000, y-axis: distance from the centre of cluster* . . . 32
4.7 *Isolation Forest Anomaly Scores, x-axis: frequency of datapoints, y-axis: anomaly score of data point* 34

5.1 *K means metrics* 37
5.2 *DBScan metrics* 38
5.3 *ISF metrics* 39
5.4 *LOF metrics* 39
5.5 *Histogram of metrics* 40

List of Tables

4.1 *System Configuration* 26

5.1 *Metrics Comparision* 40

List of Equations

2.1 *Cosine Similarity* 9
2.2 *k-means objective function* 9
2.3 *DB-scan distance function* 10
2.4 *Anomaly score for Isolation forest* 12
2.5 *Local outlier factor value* 12
2.6 *Accuracy* 14
2.8 *Recall* 15
2.9 *F1 score* 15

List of Abbreviations

1. KNN: K-Nearest Neighbors
2. LOF: Local Outlier Factors
3. DBSCAN: Density Based Scan Spatial Clustering of Applications with Noise
4. LRD: Local Reachability Density
5. IF: Isolation Forest
6. ANN: Artificial Neural Network
7. CNN: Convolutional Neural Network
8. IT: Information Technology industry
9. BERT: Bidirectional Encoder Representations from Transformers
10. GloVe: Global Vectorization
11. TP: True Positive
12. TN: True Negative
13. FP: False Positive
14. FN: False Negative
15. PCA: Principal Component Analysis
16. TF-IDF: Term Frequency-Inverse Document Frequency
17. BDA App: Big Data Analytics Ap-plications
18. OLS: Ordinary Least Squares
19. NLP: Natural Language Processing
20. IDF: Inverse Document Frequency
21. SVM: Support Vector Machine
22. LSTM: Long Short Term Memory
23. S-LSTM: Stacked Long Short Term Memory

24. DARPA: Defense Advanced Research Projects Agency
25. CDMC2016: Cyber Security Data Mining Competition 2016
26. RNN: Recurrent Neural Network
27. HDFS: Hadoop Distributed File System

Chapter 1

Introduction

Many large-scale Internet services run in several large server clusters. In recent days, many companies are running these services on virtualized cloud computing environments provided by several companies such as Amazon, Microsoft, and Google for scalability and pricing reasons primarily [41]. Designing, maintaining, and deploying monitoring systems for such large and complex systems is very difficult. When such services consisting of hundreds of software components running on several computers misbehave, developers and operators need every tool at their disposal to troubleshoot and diagnose problems in the system. There is one source of information that is available for almost every component of a software system that provides detailed information about the original developer's ideas about noteworthy or unusual events for monitoring and problem detection [14]. They are the good old console logs [91].

Since the dawn of programming, logging has been used by developers[19] from the simple printf to complex logging libraries, to record program variables, trace execution, report run time statistics, and even print out complete sentences to be read by a person, generally by the developer himself. However, modern large-scale services are usually developed by several developers, and the people going through the logs are part software integrators, part developers, part users, and ones are given the responsibility to fix a problem. They may not be the developers who have built the log statements. To make things worse, modern systems integrate external(often open-source) components that are frequently updated, which may change the log statements or the semantics of the log statements [81]. This further extends the dilemma for the person going through logs. Logs are usually very large files to examine manually [52, 73] and are too unstructured to analyze automatically. So, typically keywords such as "error" or "exception", are searched for but this may not be effective or enough to identify the problem [52, 73, 90].

As unusual log messages often lead to the cause of the problem, it is natural to consider log analysis as an anomaly detection problem in machine learning [44]. However, it is not always the case that the presence, absence, or frequency of a particular kind of message is enough to diagnose a problem. Usually, a problem manifests as an abnormality concerning different kinds of log messages (correlations, relative frequencies, and so on). Therefore, instead of analyzing the words in textual logs, we need to create features that precisely determine correlations among log messages and perform anomaly detection on these features [90].

An outlier is defined as an observation that deviates from other observations such that we can suspect that the system is not going through its general behavior [12]. Outlier detection can be applied to system fault detection, and unexpected error detection in databases [74]. While most outlier detection methods can be used when all the data is present at once, the need for an efficient outlier and anomaly pattern detection in a data stream has increased [35]. A data stream is a sequence of data that is continuously generated over some time. Outlier detection is generally done on individual samples to predict outliers in the sample. In contrast, anomaly pattern detection on a data stream involves detecting a time point where the behavior of the system is not usual and significantly different from the general behavior [44, 74].

Anomaly detection in real-time streaming data has significant applications in a wide variety of industries involved in activities such as preventive maintenance, fraud detection, and monitoring. These activities can be found in many industries like finance, IT, security, medical, energy, e-commerce, and social media. An anomaly does not necessarily mean a problem. It implies a change in the behavior of the system. A change might be negative, such as a bug in the system, or positive, such as a better performance of a particular software component after an update, which could be further reviewed after identifying it. Anomaly detection in streaming applications is particularly challenging since there could be several data stream sources. So, there is very little scope for a human expert to intervene. Thus, there is a necessity for automation of detection of anomalies to identify and adapt to the changes and draw prediction. This automation helps the developers hugely in the log analysis [13].

At Volvo Group IT, the teams develop and maintain several applications need automation of log analysis. The anomaly detection in logs could help them by identifying the changes in the systems, troubleshoot and detect problems. Thus, we propose this research to identify the best machine learning algorithms for anomaly detection in logs for the automation of log analysis. In this thesis, we have done a literature review to identify some unsupervised learning algorithms that can be used for the detection of anomalies. We have done an experiment involving log parsing, feature extraction using algorithms such as GloVe, and application of algorithms such as K-means, Isolation Forest, DB Scan, and Local Outlier Factor for anomaly detection. We have found that Local Outlier Detection performed significantly better than algorithms based on most metrics considered.

1.1 Aim

The aim for this research is to identify different state-of-the-art algorithms for anomaly detection in log files through a literature review and to compare them and provide results. Different metrics like accuracy, precision, recall and F1-score are used for comparing the algorithms.

1.2 Objectives

For achieving the aim of the thesis, we have drawn the following objectives:

Objective 1: To identify different state-of-the-art methods for anomaly detection in log files through a literature review.

Objective 2: To compare the identified methods by a literature review and identify the best among them.

Objective 3: To implement the algorithms identified in the literature review.

Objective 4: Comparison among the identified algorithms.

Objective 5: Finding out the best algorithm among the compared algorithms for anomaly detection in logs and present the results.

1.3 Research questions

To reach the objectives of the research, the following research questions have been formulated:

RQ1: What are the different state-of-the-art methods or algorithms that could be used for anomaly detection in logs?
This question is answered by doing a detailed literature review.

RQ2: Which algorithm results in best performance among the algorithms identified through RQ1?
An experiment is done to answer this question to identify the best algorithm for anomaly detection in logs.

1.4 Defining the scope of the thesis

This research focuses on analysing anomalies in the log files using different state-of-the-art machine learning algorithms. A comparison among the algorithms is done, where the algorithm that gives the best performance metrics is identified. This study will not be effecting any environmental factors.

1.5 Outline

The thesis structure is divided into seven different chapters which are as follows:

- **Chapter 1:** This chapter contains the introduction of this thesis, aim and objectives, research questions, and motivation.

- **Chapter 2:** In this chapter, we discuss the background and the concepts used during our thesis.
- **Chapter 3:** This chapter contains the summary of the works related to this thesis.
- **Chapter 4:** This contains methods to answer research questions. It includes the method for literature review and the experiment.
- **Chapter 5:** Results obtained are presented in this chapter.
- **Chapter 6:** This chapter consists of analysis and discussions about the results and methods, the contribution of the thesis to the existing research, threats to the validity of the thesis.
- **Chapter 7:** In this chapter,we discuss the conclusion of the thesis and discussion on possible future work.

Chapter 2

Background

2.1 Anomaly Detection

Anomaly detection is a vital problem and plays a very crucial role in many application domains and has been researched within diverse research domains [28]. Anomaly detection specifies to identify rare or interesting patterns in the given data [20]. The identified patterns here are known as anomalies, outliers, exceptions, peculiarities in various application domains. The applications that use anomaly detection are credit card fraud, healthcare, cybersecurity, military surveillance, etc. [28]. There are several challenging events that anomaly detection encounters with, which are:

1. The anomalies that result due to malicious actions, often appear to be normal, so distinguishing them is difficult.

2. Anomaly detection varies in various application domains, the same notion for an application is different with the same fluctuations. Therefore, the technique applied in one domain varies from the another.

3. Here, the availability of the labeled data for training the model is difficult.

4. The noise in the data is difficult to distinguish from the anomalies identified.

All the above challenges determine that anomaly detection is not easy and it is resolved using unsupervised anomaly detection [28]. Unsupervised anomaly detection is an anomaly detection that overcomes the challenges faced by anomaly detection algorithm. There are two assumptions made by the unsupervised algorithm:

- Assumption 1: The normal instances form larger clusters when compared to the intrusion instances [79].

- Assumption 2: Since the intrusions and normal instances are qualitatively different, these both do not fall into same cluster category [79].

This algorithm takes unlabelled data as an input data-set that means to find intrusions or abnormalities buried within them. After the detection of these abnormalities, the data can be used to train a traditional anomaly detection algorithm or misuse detection algorithm [56].

2.2 Machine Learning

Machine Learning is a set of methods that can automatically detect patterns in data and then use the uncovered patterns to predict future data or to perform other kinds of decision making under uncertainty [17]. It is a part (subset) of Artificial Intelligence. For a system to be intelligent, it should have the ability to learn and adapt to the changing environment. Some fields where machine learning can be used are text classification, Natural Language Processing (NLP), computer vision applications, network intrusion, medical diagnosis, military, space equipment, etc. [68]. It is divided into different types such as

- Supervised Machine Learning
- Unsupervised Machine Learning
- Semi-Supervised Machine Learning
- Reinforcement Machine Learning

2.2.1 Supervised Machine Learning

In predictive or supervised learning, the data given as the input is labeled data which is called features, attributes, or co-variates. Here, the output data is also known for the model to learn. The scenarios that involve supervised learning are classification and regression problems.An example for supervised learning is the spam detection problem [68].

Classification This consists of assigning a category to each item. For example, document classification consists of assigning a category such as politics, business, and sports to each document. Likewise, image classification consists of assigning to each image [68].

Regression Regression is just like classification but except the response variable, it is continuous [69]. Examples for regression are predicting stock market, predicting the temperature at any location. The types of popular regression techniques are:

- Linear regression.
- Logical regression.

2.2.2 Unsupervised Machine Learning

In unsupervised learning, the data given to the model is the unlabelled data. It makes predictions for all the unknown errors [68]. This is the most widely used and applicable for the real-time complex models as there would be no requirement of the human to manually label the data. Examples of unsupervised learning involve clustering, dimensionality reduction, etc. [68].

Clustering involves the partitioning of the set of items into a homogeneous subset. It is mainly used when there are large datasets. In clustering, our first goal is to

estimate the distribution over the number of clusters. The second goal is to estimate which cluster each point belongs to [69]. The data points that represent the same group show similar features/ properties unlike the data points in different groups that show dissimilar points are known as clusters. It is an unsupervised learning technique [82]. These clustering methods are then categorized as

1. **Partitional methods:**
   Partitional algorithms are the clustering algorithms where the datasets are divided into k number of groups which are pre-specified by the analyst. There are different types of partitional algorithms such as K-means clustering, K-medoids clustering and so on [16].

2. **Hierarchical methods:**
   It is an alternative approach for partitional clustering methods. This does not require any specified number of clusters that needs to be generated. The Hierarchical clustering results in a tree-based which is also known as dendrogram [16].

3. **Density-based methods:** Density-based clustering algorithms play an important role in finding shapes in the data when it is density- based. Different types of algorithms are created using this method such that if clusters are created using the density of neighborhood objects then the DBSCAN algorithm is used or if clusters are created according to a density function then DENCLUE is used [85].

4. **Grid-based methods:** The grid-based clustering algorithm does not concern with the data points but with the value space that surrounds the data points[42]. Examples for grid-based clustering method is STING (STatistical INformation Grid-based clustering method) [42].

2.2.3 Semi-Supervised Machine Learning

In Semi-supervised learning, the input given is both labeled and unlabelled data and then makes the predictions for all the unknown points. Semi-supervised learning is used where the unlabelled data is available easily and the labeled data is not. It is mainly used for classification, regression, clustering, and ranking tasks [68].

2.2.4 Reinforcement Machine Learning

In reinforcement learning, the training and testing phases are both combined. In Machine learning, there are series of actions received as the output for its applications. No single action can be considered as the desired output but the sequence of actions to reach the goal. Therefore, in machine learning, these good action sequences are able to generate a policy [17]. This learning is mostly used in game playing, where a single good move will not result in a good game play, but a sequence of good moves will be helpful in a good game play. It consists of environmental states, a set of scalar signals, and a discrete set of agent actions. For example, Chess is played with a very less number of rules but is a very complex game comprising of different set of

moves at every stage of the game. But, with the learning, the algorithm is made in such a way that it is efficient and adaptable to its environment [17].

2.3 Word embeddings

Word embeddings are the representation of words in the form of numerical vectors. These are used in various natural language processing applications. Primarily, they enable the computation of semantically related words. Also, they can be used to represent other linguistic units such as phrases and short texts, reducing the inherent sparsity of traditional vector-space representations [76, 81]. It is defined as the collection of language modeling and feature learning techniques where words or phrases from vocabulary are mapped to vectors of real numbers [81]. There are different methods used for different tasks and each one is better depending on its particular task. The different uses are Parameter learning, Word representations, Hierarchical representations, latent semantic indexing [76]. There are several word embedding techniques such as word2vec [66], BERT [32] and GloVe [77].

2.4 Algorithms

The algorithms used in this thesis are mentioned below:

2.4.1 GloVe:

GloVe, Global vectors for word representations is an unsupervised machine learning algorithm for words to be represented as vectors [78]. This is a method for obtaining pre-trained word embeddings. The GloVe is mainly used for two things:

- Creating word vectors that carry the same meaning in the vector space.
- It considers global count statistics not only the local information.

It learns on the co-occurrence matrix and trains word vectors so that the co-occurrence ratio can be predicted. In comparison with word2vec, GloVe is very similar, despite their starting points. Mikolov et al. introduced a new scheme based on word analogies that explores the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference [66]. For example, let us consider an analogy "the king is to queen as man is to woman" when decoded with the vector equation in the vector space, king queen = man woman [78].

2.4.2 Cosine similarity

The similarity when two documents are considered, where these two documents are to be term vectors, corresponds to the correlation between the two vectors. This is the cosine similarity [48]. For the application of cosine similarity, word count vectors must be calculated with the use of vectorizers. It can also be defined as the measurement that quantifies the similarity between two or more vectors. It is the

cosine of the angle between the two vectors [15]. It is applied in clustering and also numerous information retrieval applications. Cosine similarity's important property is that it does not depend on the document length. For example, when we consider two documents,d and d_1, where d_1 is an identical copy of document d. The cosine similarity for d and d_1 is one. Hence, we consider both the documents to be identical. This means, that the documents with the same composition and different weights are said to be identical [48]. Therefore, Given two documents $\overrightarrow{t_a}$ and $\overrightarrow{t_b}$, their cosine similarity is

$$SIM_c(t_a, t_b) = \frac{\overrightarrow{t_a} \cdot \overrightarrow{t_b}}{|\overrightarrow{t_a}| \times |\overrightarrow{t_b}|} \tag{2.1}$$

where $\overrightarrow{t}_a$ and $\overrightarrow{t}_b$ are m-dimensional vectors over the term set T = t_1, . . . , t_m. Each dimension represents a term with its weight in the document, which is non-negative. As a result,the cosine similarity is non-negative and bounded between 0 and 1 [48]. The above mathematical equation can be described as the division between the dot product of vectors and the product of the magnitude of each vector [15].

2.4.3 K-means-algorithm

K-means algorithm is a clustering algorithm. It is an unsupervised technique. These algorithms are best suited for handling large datasets as the computational requirement is relatively low. K-means is an iterative partitional clustering algorithm [88]. When compared to the traditional supervised machine learning algorithms, this clustering algorithm classifies data without having trained it first with labeled data. Some applications of K-Means in real-world are computer vision, search engines and market segmentation [64]. K-means processes the learning data in such a way that it first selects random points known as centroids, which are used as the starting points of each cluster, and for positioning of the centroids it performs some repetitive calculations. This iteration is repeated until there are no centroids left in the cluster [40]. The K-means algorithm works with minimizing the distance between the clusters. Because of this reason, the similarity measures cannot be directly used by the algorithm. The algorithm consists of two phases, where the first phase selects the k centers randomly when the value is fixed and the second phase is to take each data point closest to the center of the cluster [70]. Euclidean distance is defined as the distance measure that determines the distance between each data object and cluster center. Let the target object be considered as x and x_i which indicates the average cluster C_i which is the criterion function,

$$E = \sum_{i=1}^{k} \sum_{x \epsilon C_i} |x - x_i|^2 \tag{2.2}$$

Where, E is the sum of the squared error of all object, x and x_i are the target objects. k is the number of cases, C_i is the average number of clusters, x is the case i and x_i is the centroid for cluster k. $|x - x_i|^2$ is the distance function [70].

2.4.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering machine learning algorithm. It is an unsupervised method. It is specifically used to discover clusters and noise in a spatial database. Here, when at least one of the parameters and one point from each respective cluster is known, then the retrieval of all points that are density reachable from the assigned point using the correct parameters is performed [36]. The actual idea behind DBSCAN is to find the regions of high-density and low-density [84]. This algorithm is very efficient for large spatial databases. In this algorithm, there are two important input parameters EPS and MinPts where EPS is defined as

$$Eps(p) = \{q \epsilon D | distance(p, q) \leq Eps\} \tag{2.3}$$

MinPts is defined as the minimum number of points required to form a cluster. DBSCAN requires both EPS and MinPts. It works in such a way that it starts with an arbitrary starting point p that has not been visited and it retrieves all the points that are density reachable from p [84]. If this does not take place, then the point is noted as noise. When a point is considered to be a part of a cluster, then its neighborhood is also considered to be a part of the cluster [36]. Later, all the points are added to the neighborhood as if their own. If suppose p is not density reachable, then DBSCAN visits the next point in the database which leads to the discovery of the cluster or noise [84].

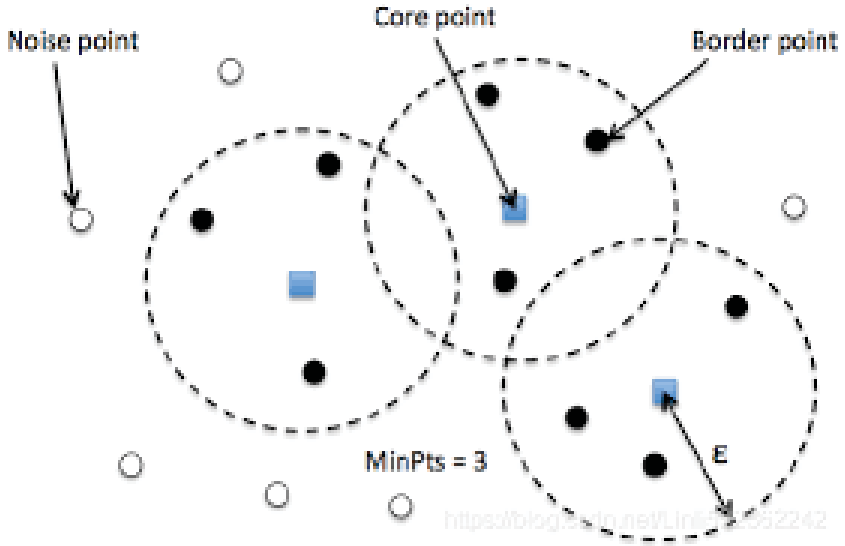

Figure 2.1: *DBSCAN*[84]

Where, p and q are arbitrary starting points in the database D. Eps (ϵ) is the distance from the point to locate the neighboring points.
Here, the core points, border points, noise points are explained:

1. Core Points: In the surrounding region of ϵ, if the point contains at least min-points number of points, then these points are called core points [84].

2. Border Points: In the surrounding region of ϵ, the points that have less than min-points number of points are called the border points [84].

3. Noise Points: In the surrounding region of ϵ, the points that are not core, border points and have less than min-points number of points are called noise points [84].

2.4.5 Isolation Forest

Isolation forest is an unsupervised algorithm, that builds an ensemble of isolative trees that consists of multiple itrees in the dataset. The main idea behind this outlier detection method is that Isolation forest identifies anomalies but not the data points [57]. The concept is based on decision trees. Here, we create partitions by first selecting a feature randomly and then randomly selecting a split value between the minimum and maximum value of the selected feature [57]. The main principle is that the outliers identified are less frequent than the usual observations and are also different from them in terms of values [57]. Here, the anomalies are the instances that have short average path lengths on isolation trees. There are two variables, which are [62]

1. number of trees required to build

2. sub-sampling size

In figure 2.2, shows the representation of the iForest where it is shown how the itree is varied. The scores here represent the presence of the outlier where the color red is the outliers, dark blue gives us the common spaces and light blue is for uncommon spaces. Here, isolation means the separation of an instance from the rest of the

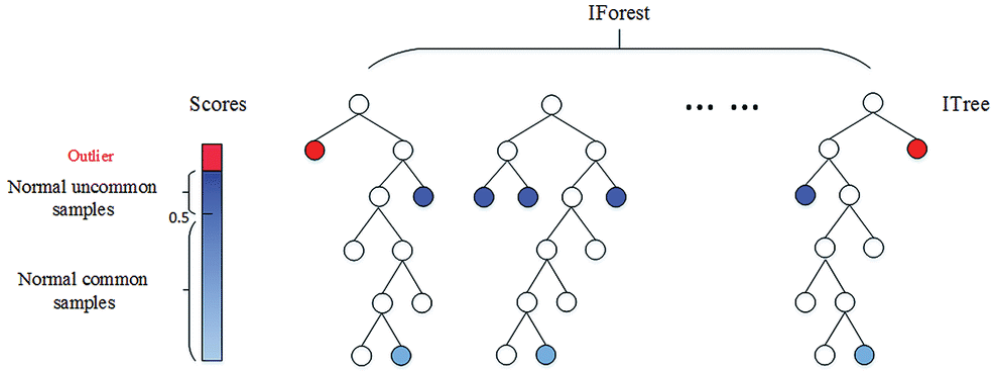

Figure 2.2: *IsolationForest*[31]

instances. Since there are few anomalies they can be differentiated and separated easily. The process of partitioning in random trees of data instances is done until

all the instances are separated. This process produces a shorter path for anomalies or outliers due to shorter paths in trees and instances with different attributes are partitioned in early stages [62]. Isolation forest depicts that anomalies are more susceptible to isolation under random partitioning [62] and also that the average path lengths converge when the number of trees increases [62]. These isolation trees are created by randomly choosing attributes and their split values and also the minimum and the maximum values of the selected attributes. At every node of the isolation trees, the instances are divided into two parts basing on the attributes and their values. For anomalous instances, the attribute values are very different when compared to the normal instances as the attribute values can be easily divided from them. The average depth of the instance in the isolation forest is calculated for alleviating the effects of the randomly chosen values and to use it as an anomalous score of the instance. The lower score of the instance here determines that the probability of an anomaly is high [34]. The anomaly score in this case for Isolation forest is defined as:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2.4}$$

where 's' is the anomaly score, 'n' is the number of external nodes, h(x) is the path length of observation, 'x' denotes an instance, c(n) is the average length of the path for n instances, E(h(x)) is the average length of the path from the isolation trees [62].

.

2.4.6 Local outlier factor

The Local outlier factor (LOF) is a clustering algorithm. This algorithm is used to identify clusters present in the dataset. When a point which is known to be an outlier present in its local neighborhood, it is known as a local outlier. It is a density-based clustering approach [26]. In LOF, two concepts needs to be discussed such as LRD (Local Reachability Density) and KNN (K-Nearest Neighbors) for better understanding. LRD tells us how far a point is from the nearest cluster of points. K-distance is the distance between the point, and its Kth nearest neighbor. KNN is the set of points that lie in or on the circle of radius K-distance. Local Outlier Factor is the ratio of the average LRD of the K neighbors of a point A to the LRD of point A [15]. Local outlier factor is used to measure the degree of small category from a large category, that is, it determines the patterns that are represented by small category deviate from the normal or legitimate patterns where the LOF value is determined by the number of components in the clusters and its distance to the nearest local category [39].

When a point o is given, in the dataset, the value of the LOF is as follows:

$$LOF(O) = |c_i| \times min[distance(o, c_i)] \tag{2.5}$$

Where $o \in c_i, c_i \in SC, c_j \in LC$. When the LOF value is higher, the farther the point o deviates from the normal behavioral patterns. When the LOF value is fixed for each object, we get to know the actual behavior of the patterns in the given

dataset. The clusters are divided into large clusters (LC) and small clusters (SC). For the clustering result of dataset D, let C= $\{c_1, c_2, c_3...c_k\}$ [39].

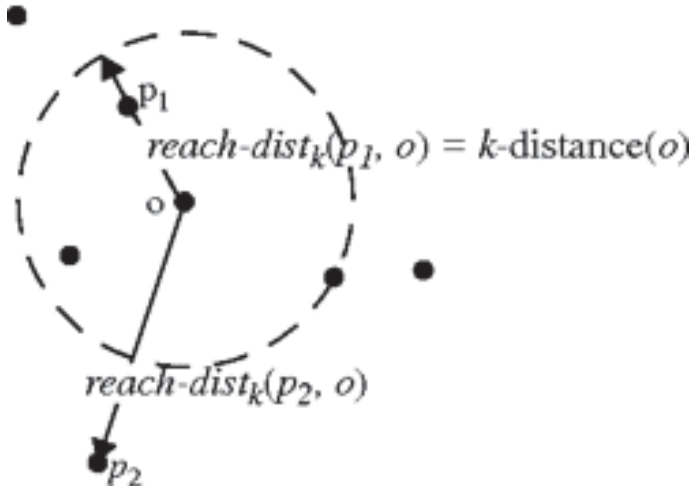

Figure 2.3: *Local Outlier Factor*[25]

In Figure 2.3, where k is a natural number, the reachability distance of an object with respect to object o. In figure 2.3, k is taken to be 4. It can be seen that if the object p_2 is far away from o, the reachability distance between the two is their actual distance. However, if the object p_1 is close to 'o', then the actual distance is replaced by the k-distance of 'o'. The higher the value of k, the more similar the reachability distances for objects within the same neighborhood. In a density-based clustering algorithm, two parameters needs to be considered for defining density,

1. MinPts, which is the minimum number of objects.
2. That specifies the volume

These two parameters in the clustering algoirthms operate by determing the density threshold and for the detection of density-based outliers, it is necessary to compare the densities of different sets of objects [26].

2.4.7 k-Nearest Neighbour (KNN) Algorithm

The k-nearest neighbors algorithm is the classification algorithm. It is typically used for calculating the predefined distance of an unknown sample in the training data set. It is known for the advantages such as [21]

1. its inherent simplicity
2. its robustness to noisy training data
3. its effectiveness if the training data is large.

Even with its advantages, the KNN algorithm depends on the following factors [21].

1. The optimum value of the parameter k (the number of nearest neighbors).
2. Choice of a proper distance measure [75].
3. Selection of an appropriate similarity measure [67].

The KNN rule classifies x by assigning it as the label that is most frequently represented among the K nearest samples; this means that a decision is made by examining the labels on the K-nearest neighbors and taking a vote [75]. KNN classification was developed by the requirement to perform discriminant analysis when reliable parametric estimates of probability densities are unknown or difficult to determine [75].

2.5 Performance metrics

Performance metrics are used in machine learning for evaluating different machine learning algorithms. The different performance metrics used are precision, recall, accuracy, F1 score, training time, prediction time. Let us define the terms associated with the metrics,

- True Positives (TP) The case where both the actual class predicted class of data point is 1.
- True Negatives (TN) The case where both the actual class predicted class of data point is 0.
- False Positives (FP) The case where an actual class of data point is 0 the predicted class of data point is 1.
- False Negatives (FN) The case where an actual class of data point is 1 the predicted class of data point is 0.

2.5.1 Accuracy

Accuracy is defined as the number of correct predictions made by a total number of the predictions made.

$$\text{Accuracy} = \frac{TP + TN}{(TP + FP + FN + TN)} \tag{2.6}$$

where, TP is True positive, TN is True negatives, FP is False positives, FN is False negatives

2.5.2 Precision

Precision is defined as the measure of correct value given by the machine learning model where TP and FN are used. It is the ratio of True Positives to the total positive cases predicted.

$$\text{Precision} = \frac{TP}{(TP + FP)} \tag{2.7}$$

where, TP is True Positive, FP is False Positives

2.5.3 Recall

Recall is defined as the number of positive values returned by the machine learning model where TP and FN are used.

$$\text{Recall} = \frac{TP}{(TP + FN)} \tag{2.8}$$

where, TP is True Positive, FP is False Negatives

2.5.4 F1 score

The F1 score represents both precision and recall. The weighted average of Precision and Recall.

$$\text{F1 Score} = \frac{2 \times (Recall \times Precision)}{(Recall + Precision)} \tag{2.9}$$

2.5.5 Run time

The time taken for the algorithm to detect anomalies in the logs.

Chapter 3

Related Work

Logs are widely used for analysis and trouble shooting of problems. There has been several studies on anomaly detection for log analysis. There have been mainly statistical methods, supervised learning algorithms, unsupervised learning algorithms and deep learning algorithms used for anomaly detection for log analysis.

Qiang Fu et al. in this research paper [38] described the importance to automate log analysis as the complexity of the distributed system continuously increases and they have used FSA(finite state automaton) model to learn the log sequences, using the algorithm proposed by Mariani et al. [65], and then to detect anomalies using statistical analysis. The data used for log analysis is unstructured data.

Peter Bodìk et al. [22] have used statistical methods to identify and classify crises for the system, specifically which have been seen before, so that a know solution can be applied easily. They have also used logistic regression algorithms for extracting metrics from logs obtained from hundreds of machines running in an enterprise level database. Using these metrics, crises were classified and identified.

Chen et al. [30] have used a supervised learning algorithm using labelled data. Decision tree learning approach is used to identify the failures using logs of eBay Internet service system. The features used for Decision tree are extracted from logs.

Liang et al. [60] have also implemented supervised algorithms along with other algorithms for anomaly detection using IBM BlueGene/L logs, however they have used SVM (Support Vector Machine) algorithm for training and testing data. Several other algorithms such as RIPPER (a rule based algorithm), Nearest Neighbour based algorithms have been implemented to compare the performances of the algorithms.

Mostafa Farshchi et al. [37] have also implemented a supervised learning algorithm for anomaly detection in logs from Amazon Ec2 computers [2] while running tools such as Netflix Asgard [4] and CloudWatch [1]. However, a Multiple Regression technique, called Ordinary Least Squares (OLS) regression has been adopted to correlate system failures with log events. A statistical method called Pearson product-moment correlation coefficient has also been used to identify correlation between different log events and cluster them into groups called activities [50].

All the above mentioned supervised algorithms are able to learn patterns in the

data, consequently performing better than unsupervised algorithms. However, it is time-consuming to label large amounts of data for training these models. Moreover, they will not be able to identify new patterns of anomalies which they are not trained of. This is of most essential need in identifying system failures for software systems since most of the historical anomalies are fixed and tried to be not repeated. Identifying new types of anomalies is essential to prevent system failures.

Xu et al. [90] have applied PCA (Principal Component Analysis) algorithm which is an unsupervised learning algorithm to detect anomalies in the logs. The dataset they have used are from Hadoop Data File System (HDFS) [3] and Darkstar online game server [11]. The logs have been parsed and features such as log message count vectors were extracted from the logs. These vectors were used to do PCA and the results were visualized using decision trees.

Weixi Li [58] extracted features from logs using Natural Language Processing (NLP) techniques such as TF-IDF (Term Frequency-Inverse Document Frequency). And then unsupervised learning algorithms K-means and DB-Scan were used in combination with Artificial Neural Network (ANN). Li has recommended a solution of using features of time stamps and character bigrams and a modified version of K-means algorithm in combination with an ANN (Artificial Neural Network).

In this study [83], Shang et al. have clustered logs by extracting information such as ids, time stamps, etc. and linked them using features such session ids. They have clustered the logs into patterns they occur. So, when an anomalous pattern occurs, it could be easily identified. The logs collected are from Big Data Analytics Applications (BDA App) which are used for analyzing big data using massive parallel processing frameworks such as Hadoop.

Hamooni et al. [43] have also used an unsupervised algorithm called LogMine which involves clustering of the logs based on tokens generated from logs by parsing the logs based on general structure of the logs. This was an extension of studies done by Ning et al. who have implemented DBScan and a modification of DBScan algorithm for log analysis [71]. Hamooni et al. have used six datasets of logs for their studies.

Lin et al. [61] have taken an unsupervised learning algorithm approach using clustering called LogCluster. Logs from Hadoop-based applications were collected and used for this study. The logs have been parsed and vectorized using NLP (Natural Language Processing) techniques such as IDF (Inverse Document Frequency) and Contrast-based Event Weighting. These vectors were used to cluster the algorithms. The performance obtained by the LogCluster algorithm was better and outperformed the model proposed in [83].

Lou et al. [63] have also made use of an unsupervised learning method called in variants mining. This involves parsing the logs into parameters which are further used to group the logs into log message groups and log counts for each type of log message group are used to solve a linear equation with log counts of each log message

as variables. This equation is used to identify patterns which are anomalous. The method has been tested on Hadoop [3] based systems developed by Microsoft.

Shilin He et al. [47] in this research paper have reviewed in detail, the generally used anomaly detection methods for automation of log analysis and then evaluated the methods based on their performance. A detailed review and evaluation of six state-of-the-art log-based anomaly detection methods, including three supervised methods, namely logistic regression [22], decision tree [30] and SVM (Support Vector Machine) [60], and three unsupervised methods, namely Log Clustering [61], PCA (Principal Component Analysis) [90] and In-variants Mining [63], has been conducted and further evaluated these methods on log datasets based on F-measure and run time, HDFS data set used in [90] and BGL dataset used in [73]. In this study, invariant mining outperformed other algorithms in terms of F-measure.

Brier et al. [23] introduced a dynamic rule based anomaly detection method for identifying security breaches in log records. This system is used in combination with clustering methods. They use clustering methods (unsupervised learning methods) to link logs that are related and call them transactions, logs which are clustered in transactions are checked for conditions satisfying an anomaly profile. If the log satisfies the conditions, the record is used to create a rule. These rules are made from training data to identify anomalous logs and are applied directly to identify anomalous behaviours in the systems. For testing purposes, two datasets were used 1998 DARPA (Defense Advanced Research Projects Agency) [6] Intrusion Detection Evaluation Set, which was made by monitoring a system for two weeks and Snort logs [10], created by analyzing the DARPA data set.

Vinay Kumar et al. [87] has taken a deep learning approach to detect anomalies in logs. The dataset used was provided by Cyber Security Data Mining Competition (CDMC2016). The have concluded that a S-LSTM (Stacked Long short-term memory) network architecture achieved a highest accuracy of 0.996 with false positive rate of 0.02 on the provided test dataset by CDMC2016. A S-LSTM network in an RNN (Recurrent Neural Network) layer over an LSTM.

Silin He et al. [46] have used NLP (Natural Language Processing) technique IDF (Inverse Document Frequency) weighting to vectorize the log sequences and then take the sum of all the words of the sequence to cluster the logs into patterns. The clustering algorithm is called cascading clustering algorithm which utilizes Hierarchical Agglomerative Clustering to find out the centroids of the clusters over a period of time. These logs over different times are analysed to find correlations and the most impactful patterns. This process is named Log3C and found to have better results compared to PCA (Principal Component Analysis) done by Xu et al. [90] and invariants mining done by Lou et al. [63].

Zhang et al. [93] proposed a new algorithm LOGROBUST based on natural language processing (NLP) techniques and deep learning algorithms. Like Xu et al. [90], log count feature vector has been made use of but the words were vectorized using Fast text algorithm [53] which also takes in semantics of the words. Then these fea-

tures are used to trained a Bidirectional LSTM (Long Short Term Memory) network as proposed by Huang et al. [49]. This model has outperformed models proposed in [22] (Logistic Regression), [60] (Support Vector Machine), [63] (Invariant Mining) and [90] (Principal Component Analysis).

Li et al. [59] has also used NLP techniques and deep learning algorithms for anomaly detection. The logs have been parsed to extract structured data from the logs such as block id, time stamp, etc.. The logs have been vectorized by using sentence embeddings using Bidirectional Encoder Representation from Transformers (BERT) algorithms [33]. This helps to vectorize the semantics of the log statement. All these features have been used to train an Attention Based Bidirectional LSTM (Long short term memory) algorithm which is a modification of Bidirectional LSTM network [49].

Rosenberg et al. [80] have done Spectrum-Based Log Diagnosis in their study which involves analyzing logs of system failure scenarios and system functioning scenarios, and then clustering the logs and doing statistical analysis on the logs to identify events that are strongly associated with system failures. However, the algorithm proposed does not outperform a simple dataset search by the developer analyzing logs.

As you have seen in the works [59, 93], the deep learning methods have produced really good results, however these methods are computationally expensive but also require the developer to label several logs [18] which is labour intensive since logs are ubiquitous [72]. This has been the case also at our industrial thesis sponsor, Volvo Group IT. Also, in studies [59, 93], benchmarks workloads have been used for generating logs and hand labelling them. This will not be the case in a real case scenario, and it is hugely expensive to label and also this adds a lot of complexity to the software system as explained in [18]. Also, improperly labelled data will result in bad performance for deep learning algorithms as explained in [86]. Hence, we haven chosen to experiment using Unsupervised learning algorithms in combination with Natural Language Processing techniques.

Cyntihia et al. [27] have performed a comparative analysis on algorithms Local Outlier Factor, Isolation Forest and Logistic regression for credit card fault recognition. They have found good results for both Local Outlier Detection and Isolation Forest. Krishna et al. [51] used K-means, Local Outlier Factor (LOF) and other algorithms and found good results for K-means and LOF. Breuniq et al. [24] have discussed the effectiveness of LOF in outlier detection and obtained low error rates. Kontopoulos et al. [55] have used density based algorithm, DB-Scan for identifying anomalies in logs to achieve good results. Thus, the algorithms K-means, DB-Scan, LOF and Isolation Forest have been chosen and researched in detail.

In the studies [59, 93], the researchers have chosen BERT and other algorithms for vectorization of words. We propose to choose GloVe (Global Vectors for Word Representation) [77] as it performs better than Word2Vec [66] as shown in the study [77]. Also, while BERT [32] gives the vector value for a word based on purely the

contextual meaning of the word, GloVe takes an average of all the semantics it could mean. This gives a slight advantage to GloVe since most log statements are ad hoc [29, 92], may not be grammatically correct and may be minimized to fewer words.

We have chosen to take leverage of more commonly used performance metrics such as run time, precision, recall and F-score as most of the studies discussed in this chapter have also taken advantage of these metrics.

Chapter 4

Method

Here, we give a detailed explanation of the research methodology. When we consider RQ1, a literature review was conducted for identifying the different machine learning techniques using the keywords anomaly detection, log analysis, and machine learning. Using the top-down search with the relevant keywords, the search of the required literature was done and then supplemented by the bottom-up approach. Snowballing techniques were used for the literature review. Here, we find out different machine learning techniques for anomaly detection which were used in log analysis and other domains relevant to the current research. This gives us a list of machine learning algorithms that can be used for anomaly detection, thus answering RQ1. And then an experiment is conducted for RQ2, where we compare all the machine learning algorithms, identified through RQ1, on the same data to find the best performing algorithm.

4.1 Literature Review

A literature review had been conducted for answering RQ1. The following techniques were used to conduct the literature review.

Snowballing

It refers to using the reference list of a paper or the citations to the paper to identify additional papers. However, snowballing could benefit from not only looking at the reference lists and citations, but to complement it with a systematic way of looking at where papers are referenced and where papers are cited.

The basic motivation for conducting a literature review is independent of the search approach. The initial step in the database searches is to identify keywords and formulate the search strings. Once the start set is decided then we conduct the iteration by backward and forward snowballing. As, it is important to examine the paper before finalizing it.When both the backward and forward snowballing is applied and no new papers are found the loop comes to an end, so contacting authors for potential information can be done. Now, finally, all the identified papers can go in the data extraction which is then conducted basing on the research questions [89].

According to the guidelines of Kitchenham [54], our literature review has been conducted. This consists of four major steps, which are:

- Investigation of primary studies
- Criteria for the selection of research
- Assessment of quality
- Extraction of data

4.1.1 Investigation of primary studies

The investigation of the primary studies such as performing inclusion, exclusion, quality assessment criteria from the electronic libraries, further studies were identified basing the results obtained from the primary studies, this technique is called snowballing [54]. All the articles are obtained, analyzed accordingly. For this research, several search engines such as BTH Summon, IEEEXplore, ResearchGate, Scopus, Science Direct were used. This Literature Review is used to answer research questions. Initially, in the investigation for the search of the primary studies of RQ1, the following keywords were used, anomaly detection, log analysis, and machine learning. These keywords are used for search criteria in the RQ1 and have obtained about 3000 research papers.

Search String 1:((Anomaly) AND (Detection) AND (logs) AND publication year $>$ 2012).

Search String 2: ((Automation)) AND (log) AND (analysis) AND publication year $>$ 2012).

Further, the results, which are the combination of keywords, that are generated with the use of the search string were used for the data collection.

4.1.2 Criteria for the selection of research

After the search string criteria, the inclusion and exclusion criteria of the studies for the literature review is done:

Inclusion Criteria:

- Articles in the English language
- Articles published after 2012.
- Articles that are published in electronic libraries, books and magazines.
- Articles that are related to the problem domain.
- Articles that are available in full text.

Exclusion Criteria:

- Articles that are not in English language.
- Articles that are not available in full text.

- Articles that are published before 2012.
- Articles in the form of PPT's and abstracts are not considered.

4.1.3 Assessment of quality

After the selection of the inclusion and exclusion criteria, a quality assessment is performed for the research questions by answering the questions such as

- Whether the title is related to the thesis?
- Whether the different state of the art methods are related to anomaly detection of logs?
- Whether the identified algorithm gives the best performance?

The result for these assessments is Yes or No. If the result is 'Yes', then the research shows that the study is considered useful for our thesis.

4.1.4 Extraction of data

After assessing the quality, the extraction of data is done for the research questions such as:

1. Aim of the study
2. Analysis of the different machine learning methods
3. Summary of the performance of the algorithms

This set of guidelines were followed in the research for planning conducting and reporting a systematic literature review.

4.2 Experiment

Experiment was done to compare the algorithms discussed in literature for the purpose to identify anomalies in logs. The main aim of the experiment is to test the following underlying thesis' hypotheses:

Null-Hypothesis (H_0): There are no significant differences among the algorithms' performances for the measured performance metrics.

Alternative-Hypothesis (H_1): There are significant differences among the algorithms' performances for the measured performance metrics.

The following are the dependent and independent variables for the experiment:
Independent Variables: Different algorithms being used in the experiment.
Dependent Variables: Performance metrics obtained from the experiment.

4.2.1 Experiment Set-up / Tools used

Environment:

The specifications of the hardware system used in this study is mentioned in Table 4.1. Google Cloud Virtual Machine has been used with the specified requirements to meet the memory and processing requirements for our program [5].

Table 4.1: *System Configuration*

CPU	Intel(R) Xeon(R) CPU @ 2.30GHz
Cores	4
Architecture	x86_64
System Memory (RAM)	30 Gigabytes
OS	Ubuntu 16.04.7 LTS

The programming language chosen was python since python has a clean and intuitive syntax, it is open source language and several open-source libraries are readily available for data pre-processing and machine learning algorithms.

Libraries Used:

1. **Scikit-learn:** It is an open-source library for tools for metrics, algorithms and data pre-processing [8]. It is compatible with other open-source libraries like Scipy.

2. **SciPy:** SciPy is an open-source library with algorithms and preprocessing tools such as NumPy, matplotlib, and pandas readily available [9].

3. **Apache Spark:** Apache Spark is an open source library used for processing large datasets efficiently [3].

4. **Sparknlp:** Sparknlp is a very efficient natural language processing tool which is open source, developed by John Snow Labs [7].

   In figure 4.1, you can see that console logs are taken in and undergone log parsing which involves the extraction of log variable strings and log message strings from each console log, while feature extraction involves the conversion of the variables to numerical values and extracting features such as log message type and log message count of the type from the messages after converting them to sentence embeddings using GloVe algorithm. Then these features are used for applying unsupervised algorithms to do anomaly detection which involves the last step. The Experiment involves these steps.

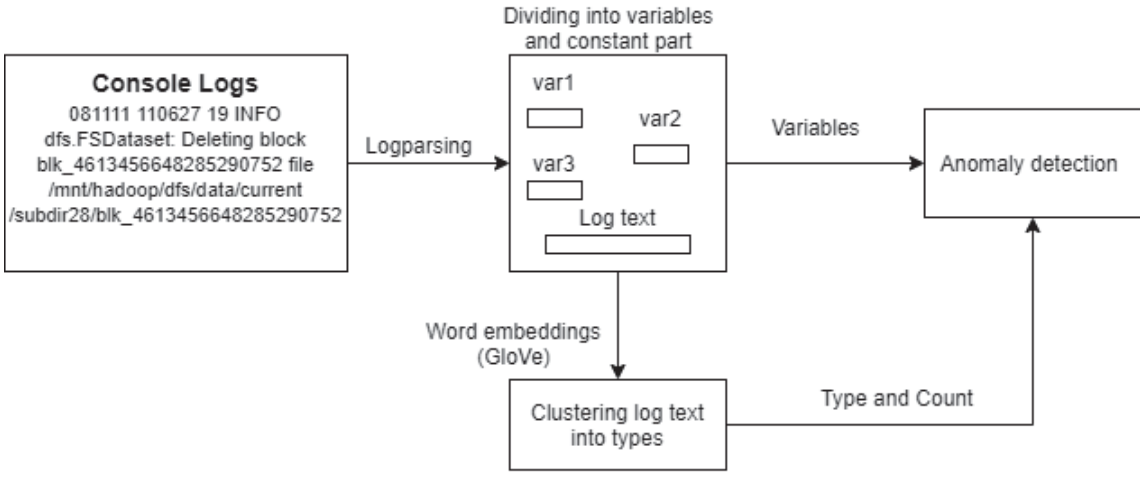

Figure 4.1: *Working method of Anomaly detection*

4.3 Data Collection

4.3.1 Dataset

Initially, we have tried to perform anomaly detection methods on logs provided by our industrial sponsor Volvo Group IT, however, due to the unavailability of labels for the logs, we have chosen an open-source data set [90] with labels for effective evaluation of the algorithms chosen. We have also tried to label the unlabelled data provided by Volvo Group IT but due to the unavailability of sufficient information and personnel for the labeling process, we had resorted to this open-source data set which has a similar structure to the logs at Volvo Group IT. HDFS (also known as Hadoop Distributed File System) is designed to run on commodity hardware. This log set is generated in a private cloud environment using benchmark workloads while running the HDFS system on a cluster of servers and manually labeled by hand-crafted rules to identify anomalies. The logs are cut into traces by block IDs. Then a ground truth is assigned to each trace associated with a specific block Id. The dataset contains 11,175,629 i.e., more than 11 million lines of console logs labeled as anomalous or not anomalous. This data set has also been used by several other studies [47, 63] along with [90].

4.3.2 Data Preprocessing

The logs file was read using spark libraries. Initially, the format of the console logs can be viewed in figure 4.2. The data pre-processing of the dataset involves two parts i.e., log parsing and features extraction which you can see in the figure 4.1.

Figure 4.2: *Before parsing the Console Logs*

4.3.3 Log Parsing

The logs are parsed using regular expressions from the console logs. The logs are parsed initially based on the general log structure and then more variables are extracted based on observation. The parsing consists of date, time, blk_id, location, log type, log message type (seen as log message info), IP address, port number and log message. The log statements after parsing can be seen in the figure 4.3.

The log statement can be divided into two parts, the variable part and the constant part.

Variable part of the logs:

The variable part of the log statement includes all values that can be unique for the log such as the date, time, blk_id, location, IP address, and port number. These values change based on the log statement, hence they are called log variables.

1. Date: It is in the format of DDMMYY (Date-Month-Year) and can be easily extracted as the first 6 characters of the log.

2. Time: It is in the format of HHMMSS (Hours-Minutes-Secs) and can be extracted as the next 6 characters of the log.

3. Block Id: It can be extracted using regex, it starts with "blk_id" and ends with a sequence of characters (numbers and symbols).

4. Location: The location is extracted using regular expressions and is a numerical value.

5. Log Type: Log type tells us what kind of message the log is conveying such as INFO, WARN, etc..

6. IP: IP address mentioned in the log statements are extracted using regular expressions. There might not be an IP address sometimes in the log statement.

7. Port: Port numbers of the IP address used by the application are also extracted using regular expressions. There might not be a port number mentioned in log statements, though the IP addresses are given.

Constant part of the logs:

The constant part of the logs contains the log message, other than the variables in the log message part, the log message remains the same for other log statements too. It is extracted using regular expressions. For example, consider the following two log messages:

"Received block blk_ -1608999687919862906 of size 91178 from /10.250.10.6"
"Received block blk_ -1608999687919862906 of size 91178 from /10.250.19.102"
It can be observed that excluding the numerical part (variable part), the remaining remains constant for both the log messages. Hence, it is called the constant part.

blk_id	timestamp	date	location	log_type	system	server_ip	ports	log_message
blk_-1608999687919862906	203518	81109	143	INFO	dfs.DataNode$DataXceiver	10.250.19.102	54106	Receiving block blk_-1608999687919862906
blk_-1608999687919862906	203518	81109	35	INFO	dfs.FSNamesystem			BLOCK* NameSystem.allocateBlock: /mnt/had
blk_-1608999687919862906	203519	81109	143	INFO	dfs.DataNode$DataXceiver	10.250.10.6	40524	Receiving block blk_-1608999687919862906
blk_-1608999687919862906	203519	81109	145	INFO	dfs.DataNode$DataXceiver	10.250.14.224	42420	Receiving block blk_-1608999687919862906
blk_-1608999687919862906	203519	81109	145	INFO	dfs.DataNode$PacketResponder			PacketResponder 1 for block blk_-16089996

Figure 4.3: *Parsed Logs*

4.3.4 Feature Extraction

Since all machine learning algorithms need numerical values for computations and predictions, the numerical features are to be extracted from the parsed log statements shown in figure 4.3. Log statements are highly correlated [90] based on different variables but grouping based on only time would result in problems [52]. So, the following are the features extracted:

1. **Block_Id:** A dictionary of block_ids is created and each block_id was assigned with a value ranging from 1 to 1008. Any new blk_id will be added to the dictionary and an incremental value will be added.

2. **Location:** Location is already a simple numerical value. Hence, it will be taken as it is.

3. **IP:** A dictionary of IP addresses is created and each IP was assigned with a value ranging from 1 to 106 in the current dataset. Any new IP address will be added to the dictionary and an incremental value will be added.

4. **Port:** Port number is also a simple numerical value. So, no further processing is required.

5. **Time:** The date and time variables in log parsing are converted into seconds and the first log statement is taken as time zero.

6. **Log Type**: A dictionary of Log Types is created and each log type was assigned with a value. The current data set has only one value, more values will be added to the dictionary on the requirement.

7. **Log Message Info:** A dictionary of Log Message Info is created and each log type was assigned with a value. The current data set has only six values, more values will be added to the dictionary based on the requirement.

8. **Log Message Type:** The log message obtained from the log parsing process is stripped of all numeral values, thus eliminating all variable values in the log message and taking only the constant part. Then, word embeddings are used

to vectorize the log statement.

GloVe: A pre-trained model of GloVe was taken using sparknlp and was further trained with 930,000 lines of logs from the data set. This helps improve the co-occurrence matrix of the GloVe model which in turn helps in better vectorization of data. The GloVe model converts each log message into a vector of 100 float points.

Classifying based on cosine similarity: A dictionary of log message types is maintained, a new entry is added if no log message type, which is stored as a vector using GloVe embeddings, in the dictionary has a cosine similarity value greater than 0.95, i.e., the log messages should match more than 95%. This approach is chosen instead of conventional clustering algorithms as the constant part will remain the same since the numerical are removed from the log message before vectorizing using GloVe embeddings.

A total of 41 different log message types had been identified in the dataset taken and stored as a dictionary.

9. **Log Count:**A dictionary for log count value for each log message type is maintained. This feature helps in distinguishing frequent log message types from rare or anomalous log message types. Hence, it is a very useful feature for anomaly detection.

All the above feature values can be seen in figure 4.4, extracted from values shown in figure 4.3. The 'log_text' in the figure is shown for referring to 'LogMesType' in the figure.

blk_id	location	ports	ips	logType	logmessageInfo	log_text	time	index	label	logMesType	logCount
1	143	54106	1	1	1	Receiving block b...	0	0	0	1	1723232
1	35	0	0	1	2	BLOCK NameSystema...	0	1	0	2	575061
1	143	40524	2	1	1	Receiving block b...	1	2	0	1	1723232
1	145	42420	3	1	1	Receiving block b...	1	3	0	1	1723232
1	145	0	0	1	3	PacketResponder ...	1	4	0	3	1706679
1	145	0	0	1	3	PacketResponder ...	1	5	0	3	1706679

Figure 4.4: *Features Extracted*

4.4 Algorithms

4.4.1 K-means algorithm

K-means algorithm is applied with one cluster and the data points outside certain threshold values are classified as anomalies. K-means was applied for 300 iterations and was run 10 times with random initial points as centroids of the data. Euclidean distance metric was used for measuring distance. The K-means algorithm was fitted with a data size of 100,000 lines. In K-means, when taken a set of data objects D and number of clusters k, some random data objects are selected to initialize k clusters, where they represent the centroid of a cluster. The rest of the data objects are then assigned to the cluster which is represented by its nearest or most similar centroid. Again, when new centroids are assigned each cluster and all the documents are re-assigned based on the centroids. The loop continues until a solution is reached, where the data objects are not re-assigned when new centroids are formed. This can be seen in algorithm 1.

The euclidean distance from the center for the data points can be observed in 4.5. For more clarity, a scatter plot was also plotted for setting the threshold values. The x-axis is scaled to a ratio of 1 to 1,000,000. From figures 4.5 and 4.6, the threshold was set at 250,000, Since, it can be seen in the histogram 4.5 that most of the points are below the distance of 0.25 (250,000 scaled).

Algorithm 1: *K-means algorithm*

Input : Preprocessed features from logs
Output: Anomalous points
Choose the number of clusters k (k = 1 in this case)
Select k random points as centroids
Assign all the points to the closest cluster centroid
Recompute the centroids of newly formed clusters
Repeat steps 3 and 4
Take the centroids
if *euclidean distance from centroid > threshold* **then**
    The point is anomalous
else
    The point is not anomalous
end

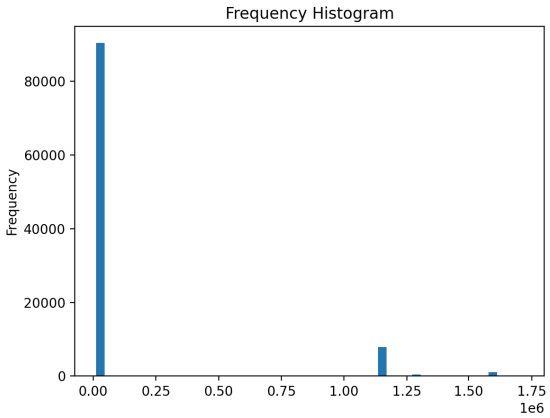

Figure 4.5: *Distribution of data over distance, x-axis: frequency of datapoints, y-axis: distance from the centre of cluster*

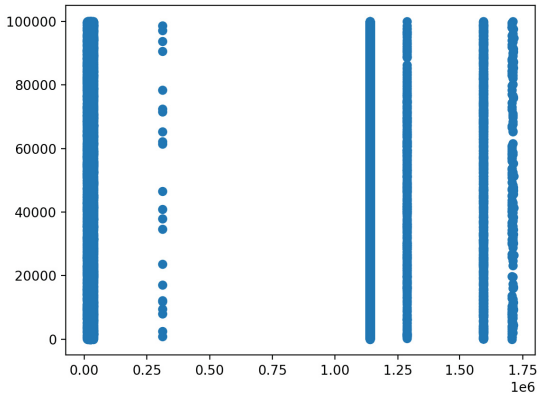

Figure 4.6: *Distribution of data over distance, x-axis: data point index value ranging from 0 to 100,000, y-axis: distance from the centre of cluster*

4.4.2 DBSCAN Algorithm

DBSCAN is applied with an epsilon value of 4 and minimum samples = 2, the distance metric used is Euclidean distance. Data size of 100,000 lines of logs was fit into the algorithm. Clusters with value '-1' are the outliers in the dataset since they could not cluster into any cluster group. MinPts is defined as the minimum number of points required to form a cluster. DBSCAN requires both EPS and MinPts. If suppose point 'p' is not density reachable, i.e., the point has less than the MinPts number of points with a distance of EPS, then DBSCAN labels the point as noise or anomaly and then visits the next point in the database which leads to the discovery of the cluster or noise [84]. In algorithm 2, P and Q are arbitrary starting points in the database D. Eps (ϵ) is the distance from the point to locate the neighboring points.

Algorithm 2: *DBScan algorithm*

Input : Preprocessed features from logs (DB), distfun, eps, minPts
Output: Anomalous points

```
1  C := 0 // cluster counter
2  for ( each point P in database DB ) {
3      if label(P) undefined then
4          continue // previously processed in inner loop
5      Neighbors N := RangeQuery(DB, distFunc, P, eps) // Find neighbors
6      if |N| < minPts // Density check
7       then
8          label(P) := Noise // Label as Noise or Anomaly
9      C := C + 1 // next cluster label
10     label(P) := C // Label initial point
11     SeedSet S := N \{P} // Neighbors to expand
12     for ( each point Q in S ) {
           // Process every seed point
13         if label(Q) = Noise then
14             label(Q) := C // Change Noise to border point
15         if label(Q) ≠ Noise then
16             continue // Previously processed (e.g., border point)
17         label(Q) := C // Label neighbor
18         Neighbors N := RangeQuery(DB, distFunc, Q, eps) // Find neighbors
19         if |N| ≥ minPts then
               // Density check (if Q is a core point)
20             S := S ∪N// Add new neighbors to seed set
21         }
22     }
23     Output points labelled as Noise which are Anomalies
24
```

4.4.3 Isolation Forest

Isolation Forest was applied on data size of 100,000 lines of logs, with all the features pre-processed. It has a contamination value of 0.1. It resulted in average anomaly scores as seen in figure 4.7. In the figure, low average anomaly scores signify more anomalous behaviors of data. These isolation trees are created by randomly choosing attributes and their split values and also the minimum and the maximum values of the selected attributes. At every node of the isolation trees, the instances are divided into two parts basing on the attributes and their values. For anomalous instances, the attribute values are very different when compared to the normal instances as the attribute values can be easily divided from them. The average depth of the instance in the isolation forest is calculated for alleviating the effects of the randomly chosen values and to use it as an anomalous score of the instance. The lower score of the instance here determines that the probability of an anomaly is high [34]. This can be seen in Algorithm 3.

Algorithm 3: *Isolation Forest algorithm*

Input : Preprocessed features from logs (DB), contamination value, isolation number or $n_estimator$

Output: Anomalous points

1 Select the point to isolate.
2 For each feature, set the range to isolate between the minimum and the maximum.
3 Choose a feature randomly.
4 Pick a value that's in the range, again randomly:// 5 and 6 are sub-steps of step 4
5 a: If the chosen value keeps the point above, switch the minimum of the range of the feature to the value.
6 b: If the chosen value keeps the point below, switch the maximum of the range of the feature to the value.
7 Repeat steps 3 & 4 until the point is isolated. That is until the point is the only one that is inside the range for all features.
8 Count how many times you have had to repeat steps 3 & 4. We call this quantity the isolation number or $n_estimator(= 50 in this case)$.
9 Repeat steps 1 to 8 till the number of steps reaches the contamination value (0.1 in this case) of the total points in the database.

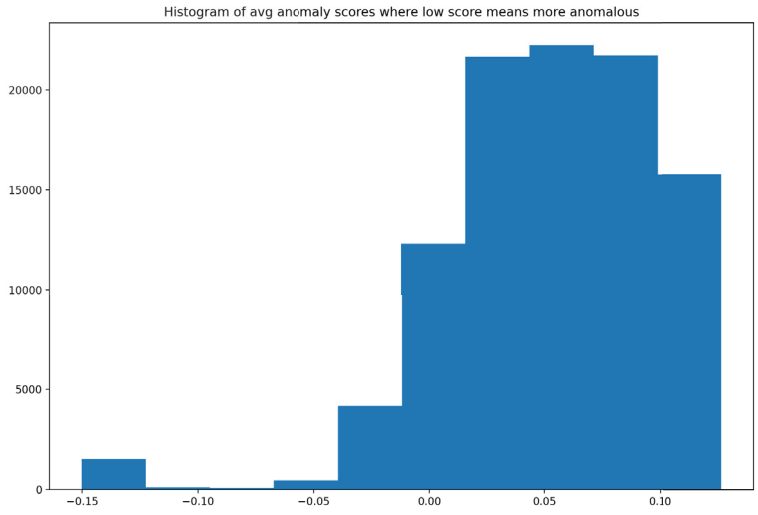

Figure 4.7: *Isolation Forest Anomaly Scores, x-axis: frequency of datapoints, y-axis: anomaly score of data point*

4.4.4 Local Outlier Factor

Local Outlier Factor(LOF) identifies anomalies by observing the neighbors of each data point. It compares the density of the total data sample with the local density of each data point. If the local density of each data point is lower, it is classified as

an outlier. KNN is applied to determine the density for the Local Outlier Factor algorithm, n=20 (which is the minimum number of neighbors) is given as a parameter for this algorithm and 100,000 lines of logs are fit into the model. Local outlier factor is used to measure the degree of small category from the large category, that is, it determines the patterns that are represented by small category deviate from the normal or legitimate patterns where the local outlier factor value is determined by the number of components in the clusters and its distance to the nearest local category [39]. In LOF, two concepts needs to be discussed such as LRD (Local Reachability Density) and KNN(K-Nearest Neighbors). LRD (Local Reachability Density) tells us how far a point is from the nearest cluster of points. K-distance is the distance between the point, and its Kth nearest neighbor. KNN (K-Nearest Neighbors) is the set of points that lie in or on the circle of radius K-distance. Local Outlier Factor is the ratio of the average LRD (Local Reachability Density) of the K neighbors of a point A to the LRD (Local Reachability Density) of point A [15]. This can be seen in the clearly explained algorithm below:

Algorithm 4: *Local Outlier factor algorithm*

Input : k (number of neighbors)
D (a set of data points)
Output: LOF values (a vector with local density factors)

```
for ( each data point p in D ){
    find k_distance, distance of kth nearest neighbour
    find N_{k_distance(p)}
    for ( each point o in k_distance ){
        reach_dist_k(p, o) = max(k_distance(o), d(p, o))}
        Calculate local reachability density (lrd_k(p))
        Compute the local outlier factor of p (LOF(p))
    }
    Outputs LOF values for all points

```

In the algorithm 4, d(p,o) is distance between p and o. The local reachability density of p is just the reciprocal of the average distance between p and points within k_distance(p). LOF(p) is the average of the ratios of the local reachability density of p and that of p's k-nearest neighbors.

4.5 Testing

The performance metrics of accuracy, precision, recall, F1-score and run time are calculated for all the algorithms using the labels available in the data after applying the algorithms. These performance metrics are measured over entire data on which the algorithms are performed since unsupervised learning algorithm are trained while running. The split of training and testing data is not done to simulate a real-time scenario, as new logs are generated, the unsupervised learning algorithms train themselves and performance increases. Validation tests such as k-fold are not performed

since the order in which logs are generated would be altered and clustering algorithms such as k-means would cluster differently when ordered differently. To avoid such cases, k-fold validation is not done. The testing is done until the performance metrics of the algorithms stopped improving in this experiment.

Chapter 5

Results and Analysis

In this chapter, the results obtained from all the algorithms performed in the experiment are discussed. The performance metrics of precision, accuracy, recall, F1-score and run time are used for evaluating the models. In the following section, the performance of each of the algorithms used are discussed and, they are presented and compared in the last section of the chapter.

5.1 K-means algorithm

In figure 5.1, performance metrics of accuracy, F1-score, precision and recall for K-means algorithm have been plotted as the records of logs are being processed. Here, it can be observed that the precision has increased rapidly and flattened with a value above 90% after about 5,000 records of logs were processed. Similarly, the F1 score was above 70%, it increased above 80 % and remained constant after processing about 7,000 records of logs. Then, the recall and accuracy for k-means started above 60% and then, were above 80% as the data increased. And they remained constant after processing about 10,000 records of logs.

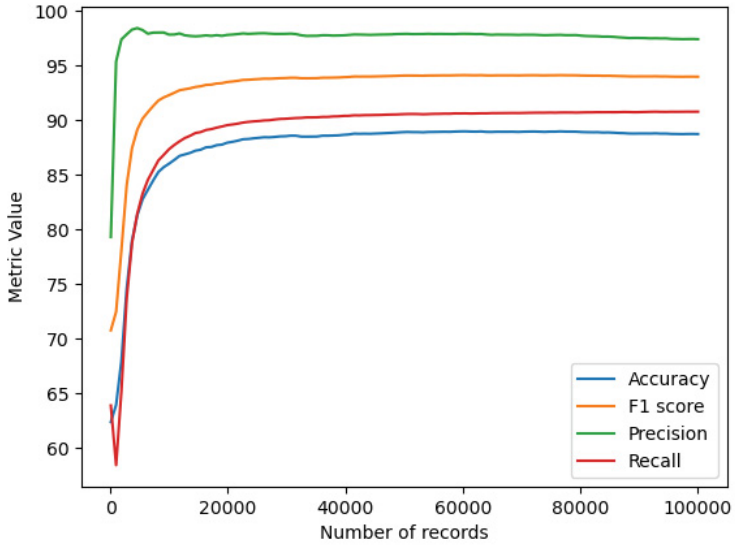

Figure 5.1: *K means metrics*

5.2 DBSCAN algorithm

In figure 5.2, performance metrics of accuracy, F1-score, precision and recall for DBSCAN algorithm have been plotted as the records of logs are being processed. Here, it can be observed that the precision has increased rapidly and flattened with a value above 90% after about 7,000 records of logs were processed. However, the F1 score was above 50%, it decreased slowly as more number of records of logs are being processed. Similarly, the recall and accuracy for DBSCAN started above 50% and then decreased slowly as more number of records of logs are being processed. This shows that DB Scan isn't reliable as the recall and accuracy are decreasing as more data is being processed.

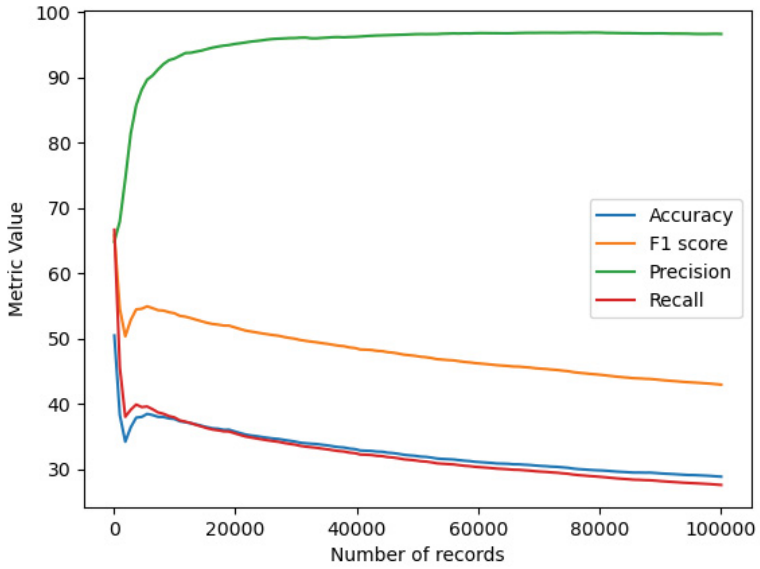

Figure 5.2: *DBScan metrics*

5.3 Isolation Forest algorithm

In figure 5.3, performance metrics of accuracy, F1-score, precision and recall for Isolation Forest algorithm have been plotted as the records of logs are being processed. Here, it can be observed that the accuracy has increased rapidly and flattened with a value above 90% after about 4,000 records of logs were processed. However, it can be observed that the F1-score, precision and recall values remained constant and very low as more data is being processed. This shows that Isolation Forest algorithm isn't reliable for anomaly detection in logs.

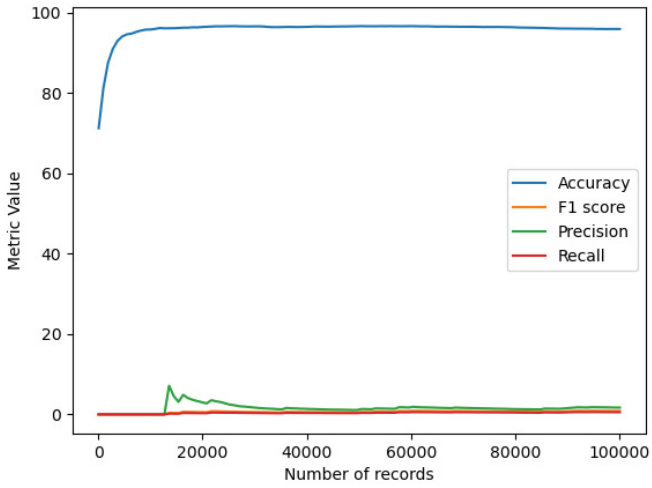

Figure 5.3: *ISF metrics*

5.4 Local Outlier Factor algorithm

In figure 5.4, performance metrics of accuracy, F1-score, precision and recall for Local Outlier Factor algorithm have been plotted as the records of logs are being processed. Here, it can be observed that the precision has increased rapidly and flattened with a value above 90% after about 4,000 records of logs were processed. Similarly, the recall, F1-score and accuracy has increased rapidly and flattened with a value above 90% after about 4,000 records of logs were processed. This shows that the algorithm is more reliable and the amount of anomalies in logs being recognized can be predicted. All the above discussed are discussed in comparative terms in the next section.

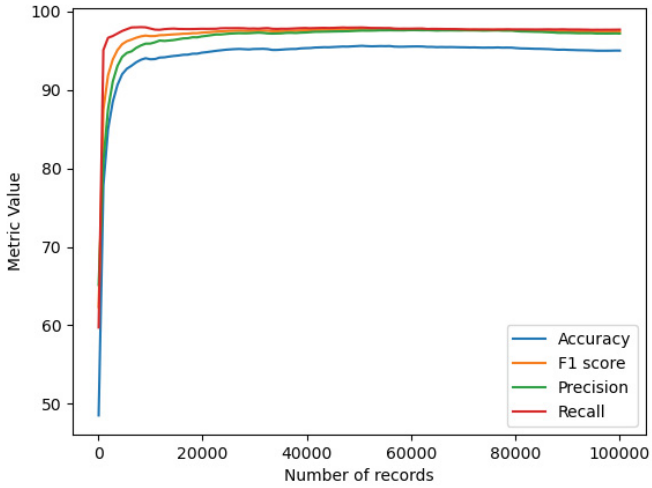

Figure 5.4: *LOF metrics*

5.5 Comparing Algorithms

Table 5.1: *Metrics Comparision*

	K-means	DBScan	Local Outlier Factor	Isolation Forest
Run time	4.1814	0.2890	5.0800	3.0400
Precision	0.97419976	0.9661402	0.97185079	0.0701
Recall Score	0.90785036	0.2763561	0.97680271	0.22983607
Accuracy Score	0.88735113	0.28903711	0.9500805	0.88352116
F1 Score	0.93985553	0.42977792	0.97432046	0.10743295

The results obtained after the experimentation can be seen in table 5.1 and figure 5.5. It can be seen in table 5.1 that there are significant differences in performance metrics of the algorithms such as Isolation Forest and K-means have a precision difference of more than 90%. Thus, the null hypothesis H_0 that all algorithms perform similarly can be rejected and the alternative hypothesis H_1 is accepted.

From the table 5.1 and figure 5.5 it can be observed that the run time for DB Scan is the least. Though, DBSCAN had a very high precision value, it has very low recall value implying that there are lot more false negatives than the positives identified correctly. Though Isolation Forest had an Accuracy score of 0.88, it had a precision of 0.07. Thus, making it a bad choice of algorithm since the low recall implies that there are lot more false negatives than the true positives identified. Due to the low recall and F1-scores, DBSCAN and Isolation Forest are ineffective for this task.

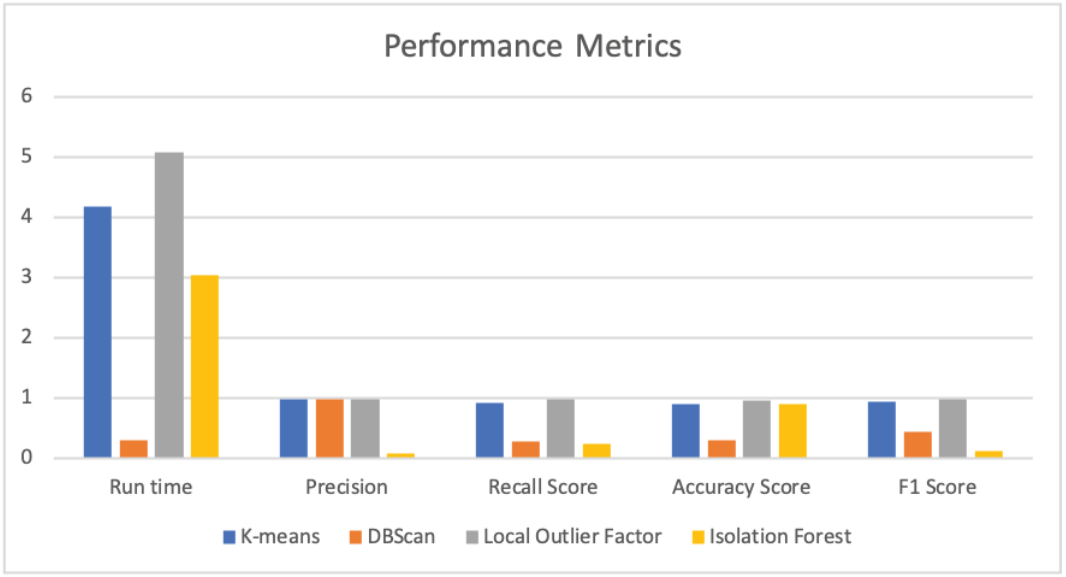

Figure 5.5: *Histogram of metrics*

K-means and Local Outlier Factor have high scores of metrics. The Local Outlier Factor algorithm has a run time of 5 secs which is slightly higher than K-means algorithm, i.e. 0.9 seconds higher. The precision of K-means is slightly higher than

that of Local Outlier Factor. However, the recall, accuracy and F1 scores of Local Outlier Factor are significantly higher. Thus, making it a better choice of algorithm since higher values of recall imply high amount of reliability.

In figure 5.5, the metrics have been compared, though to relative scale, the intention of the histogram is to visualize the relative values of different metrics of the models. And in the histogram, '0' signifies low value of the metric and the metric value is raised relative to the other values of the metric. From the histogram, it can be clearly seen that the run time of LOF is significantly higher than the other algorithms. K-means also has similar results to LOF however with lesser run time. Despite DBSCAN and Isolation forest having significantly lower run time, they have very low recall values. Hence, they can not be used for anomaly detection in logs. Hence, K-means and LOF make good choice for models. However, it depends on the user needs to choose between the two algorithms since higher run time indicates higher processing overhead and logs are ubiquitous with hundreds of thousands of lines being generated by the servers. Thus, run time makes a significant impact when processing such large datasets every second.

Chapter 6

Discussion

This study was done to identify anomalies in logs.

6.1 Answering RQ1

From the literature review, it has been observed that unsupervised learning algorithms work better for anomaly detection in logs since logs can be new as new developers take part in teams and unsupervised algorithms suit to such instances. Unsupervised algorithms K-means, Local Outlier Detection, Isolation Forest and DB Scan are some of the best algorithms that can be used for anomaly detection in logs.

6.2 Answering RQ2

In order to conduct the experiment for answering RQ2, about 11 million logs were analyzed, parsed and features were extracted using word embeddings provided by GloVe algorithm, which was trained on about a million lines of logs. The word embeddings were taken and log messages were classified log message types based on the cosine distances between the word embeddings. And then message count vector was extracted based on the log message type. All the features were converted to numerical values to fit into the unsupervised algorithms. Then, four unsupervised algorithms identified from the literature review were implemented. The four unsupervised algorithms are K-means, Local Outlier Detection, Isolation Forest and DB Scan. There significant differences in the performance metrics obtained in the experiment, hence the null hypothesis that all algorithms performed similarly was rejected and the alternative hypothesis was accepted. Though DBScan was significantly faster than all the remaining algorithms, it is not reliable. It has very low scores of recall. However, DBSCAN can be used in systems where speed is essential and False Positives wouldn't make a bad impact on the working of the system. Isolation Forest is also not effective due to its low recall values. K-means and Local Outlier Factor algorithms proved to be well performing algorithms among the algorithms identified in the literature review. Despite having a higher run time, Local Outlier Factor algorithm outperformed K-means algorithm in most performance metrics measured.

6.3 Validity Threats Analysis

Internal Validity: Internal Validity refers to how well the research has been performed. In this study, we are comparing the performance of multiple models on the same data set. Some might perform better with more data but this leads to consumption of more time and computational power. But time is an important factor since millions of logs are generated by systems and they need quick processing. So, we have tried to balance this and mitigate the problem as much as possible.

External Validity: External Validity refers to the extent to which the results obtained from the experiment can be generalized to different populations or groups i.e., different datasets and test datasets. The results might differ based on the logs selected and features extracted. There could be another way to extract features which might suit a specific algorithm.

Conclusion Validity: Conclusion Validity refers to the threats that occur when inappropriate metrics or evaluation measures are chosen and used in the experiment. This causes misinterpretation of the relationship between the dependent and independent variables.
These threats are mitigated following proper methods and steps while conducting the research. Proper metrics are applied to the evaluation of the selected algorithms, and precautionary steps were followed during the experimentation.

Chapter 7

Conclusions and Future Work

This study report discusses different algorithms that can be used for anomaly detection in logs. We have made a novel approach for feature extraction by using GloVe embeddings and using cosine similarity for classifying into types of log messages, thereby extracting important features such as log message type and log message count. We have also analyzed and parsed all the logs successfully into useful features. We have then applied unsupervised algorithms for detection of anomalies in logs. Xu et al. [90] has done log analysis with K-means and PCA, and identified that K-means works better. We have extended this work by applying anomaly detection methods on logs using Local Outlier Factor, DBSCAN and Isolation Forest. We have found that Local Outlier Factor works better than other algorithms in more than one way based on the metrics discussed. However, K-means also performed effectively with most performance metrics higher than 90% and a lower run time.

As He et al. suggest in [45] to have an automated log parsing system as there can be a wide variety of types of systems generating different formats of logs, it would be difficult to design a log parser as done in this study. Exploring different automated parsing systems would also help in further automating log analysis. Future work can be done identifying better features and more efficient ways on feature extraction. Work can also be done on experimentation with more novel and robust algorithms introduced in the future.

Bibliography

[1] "Amazon CloudWatch - Application and Infrastructure Monitoring." [Online]. Available: https://aws.amazon.com/cloudwatch/

[2] "Amazon EC2." [Online]. Available: https://aws.amazon.com/ec2/

[3] "Apache Hadoop." [Online]. Available: https://hadoop.apache.org/

[4] "Asgard." [Online]. Available: http://netflix.github.io/asgard/

[5] "Compute Engine documentation | Compute Engine Documentation." [Online]. Available: https://cloud.google.com/compute/docs

[6] "Defense Advanced Research Projects Agency." [Online]. Available: https://www.darpa.mil/

[7] "John Snow Labs | NLP & AI in Healthcare." [Online]. Available: https://www.johnsnowlabs.com/

[8] "scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation." [Online]. Available: https://scikit-learn.org/stable/

[9] "SciPy.org — SciPy.org." [Online]. Available: https://www.scipy.org/

[10] "Snort - Network Intrusion Detection & Prevention System." [Online]. Available: https://www.snort.org/

[11] "DarkstarProject/darkstar," Nov. 2020, original-date: 2014-01-11T01:31:17Z. [Online]. Available: https://github.com/DarkstarProject/darkstar

[12] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier analysis*. Springer, 2017, pp. 1–34.

[13] S. Ahmad and S. Purdy, "Real-time anomaly detection for streaming analytics," *arXiv preprint arXiv:1607.02480*, 2016.

[14] D. Ajith, "A survey on anomaly detection methods for system log data," *International Journal of Science and Research (IJSR)*, vol. 8, p. 23, 07 2019.

[15] R. Alake. Understanding cosine similarity and its application. [Online]. Available: https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a

[16] Alboukadel. Types of clustering methods: Overview and quick start r code. [Online]. Available: https://www.datanovia.com/en/blog/types-of-clustering-methods-overview-and-quick-start-r-code/

[17] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[18] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software*

Engineering and Advanced Applications (SEAA), 2018, pp. 50–59.

[19] R. Barnitz and G. Terwilliger, "Application of data-logging and programming techniques to steel mill processes," *IRE Transactions on Industrial Electronics*, pp. 24–32, 1959.

[20] I. Ben-Gal, *Outlier Detection*. Boston, MA: Springer US, 2005, pp. 131–146. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_7

[21] G. Bhattacharya, K. Ghosh, and A. S. Chowdhury, "An affinity-based new local distance function and similarity measure for knn algorithm," *Pattern Recognition Letters*, vol. 33, no. 3, pp. 356–363, 2012.

[22] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 111–124. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/1755913.1755926

[23] J. Breier and J. Branišová, "A dynamic rule creation based anomaly detection method for identifying security breaches in log records," *Wireless Personal Communications*, vol. 94, no. 3, pp. 497–511, 2017, cited By 14. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84947087413&doi=10.1007%2fs11277-015-3128-1&partnerID=40&md5=d74364bffcacd168ea9c916fc55373a0

[24] J. Breier and J. Branišová, "Anomaly detection from log files using data mining techniques," in *Information Science and Applications*. Springer, 2015, pp. 449–457.

[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[26] M. Breuniq, H.-P. Kriegel, R. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 29, no. 2, pp. 93–104, 2000.

[27] P. Caroline Cynthia and S. Thomas George, "An outlier detection approach on credit card fraud detection using machine learning: A comparative analysis on supervised and unsupervised learning," *Advances in Intelligent Systems and Computing*, vol. 1167, pp. 125–135, 2021.

[28] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[29] B. Chen and Z. M. (Jack) Jiang, "Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation," *Empirical Software Engineering*, vol. 22, no. 1, pp. 330–374, Feb. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9429-5

[30] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, 2004, pp. 36–43.

[31] W.-R. Chen, Y.-H. Yun, M. Wen, H.-M. Lu, Z.-M. Zhang, and Y.-Z. Liang, "Representative subset selection and outlier detection via isolation forest," *Analytical methods*, vol. 8, no. 39, pp. 7225–7231, 2016.

[32] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[34] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," *IFAC Proceedings Volumes*, vol. 46, no. 20, pp. 12–17, 2013.

[35] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406–421, 2018.

[36] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[37] M. Farshchi, J. Schneider, I. Weber, and J. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 24–34.

[38] 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.

[39] Z. Gao, "Application of cluster-based local outlier factor algorithm in anti-money laundering," in *2009 International Conference on Management and Service Science*. IEEE, 2009, pp. 1–4.

[40] D. M. J. Garbade. Understanding k-means clustering in machine learning. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

[41] R. L. Grossman, "The case for cloud computing," *IT professional*, vol. 11, no. 2, pp. 23–27, 2009.

[42] C. M. Guojun Gan and J. Wu. Grid-based clustering algorithms data clustering: Theory, algorithms, and applications. [Online]. Available: https://doi.org/10.1137/1.9780898718348.ch12

[43] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1573–1582. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/2983323.2983358

[44] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier detection using replicator neural networks," in *International Conference on Data Warehousing and*

Knowledge Discovery. Springer, 2002, pp. 170–180.

[45] P. He, J. Zhu, S. He, J. Li, and M. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2018, cited By 28. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056711239&doi=10.1109%2fTDSC.2017.2762673&partnerID=40&md5=7fc5baf81c30578afc82ae5a040b0bb1

[46] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 60–70. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3236024.3236083

[47] 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, 2016, pp. 207–218.

[48] A. Huang, "Similarity measures for text document clustering," in *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, vol. 4, 2008, pp. 9–56.

[49] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging," *arXiv:1508.01991 [cs]*, Aug. 2015, arXiv: 1508.01991. [Online]. Available: http://arxiv.org/abs/1508.01991

[50] R. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, 2nd ed. Melbourne, Australia: OTexts, 2018. [Online]. Available: OTexts.com/fpp2

[51] G. Jaya Krishna and V. Ravi, "Anomaly detection using modified differential evolution: An application to banking and insurance," *Advances in Intelligent Systems and Computing*, vol. 1182 AISC, pp. 102–111, 2021.

[52] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou, "Understanding customer problem troubleshooting from storage system logs," in *Proccedings of the 7th conference on File and storage technologies*, 2009, pp. 43–56.

[53] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.zip: Compressing text classification models," *arXiv:1612.03651 [cs]*, Dec. 2016, arXiv: 1612.03651. [Online]. Available: http://arxiv.org/abs/1612.03651

[54] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[55] I. Kontopoulos, I. Varlamis, and K. Tserpes, "Uncovering Hidden Concepts from AIS Data: A Network Abstraction of Maritime Traffic for Anomaly Detection," *Lecture Notes in Computer Science (including subseries Lecture Notes in Arti-*

ficial Intelligence and Lecture Notes in Bioinformatics), vol. 11889 LNAI, pp. 6–20, 2020.

[56] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, 2005, pp. 333–342.

[57] E. Lewinson. Outlier detection with isolation forest. [Online]. Available: https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

[58] W. Li, "Automatic log analysis using machine learning: awesome automatic log analysis version 2.0," 2013.

[59] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 92–103.

[60] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, 2007, pp. 583–588.

[61] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 102–111.

[62] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[63] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," 2019, pp. 231–244, cited By 38. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077114796&partnerID=40&md5=bfcee9cee3e0b98eed00361d776c69ae

[64] C. Maklin. K-means clustering python example. [Online]. Available: https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

[65] L. Mariani and M. Pezze, "Dynamic detection of cots component incompatibility," *IEEE Software*, vol. 24, no. 5, pp. 76–85, 2007.

[66] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[67] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 301–312, 2002.

[68] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[69] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[70] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *2010 Third International Symposium on intelligent information technology and security informatics*. Ieee, 2010, pp. 63–67.

[71] X. Ning, G. Jiang, H. Chen, and K. Yoshihira, "1HLAer: a System for Heterogeneous Log Analysis," 2014.

[72] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012, cited By 133. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84863395085&doi=10.1145%2f2076450.2076466&partnerID=40&md5=6ebab9cf02035e4359f15cd3b649efac

[73] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[74] C. H. Park, "Outlier and anomaly pattern detection on data streams," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 6118–6128, 2019.

[75] H. Parvin, H. Alizadeh, and B. Minaei-Bidgoli, "Mknn: Modified k-nearest neighbor," in *Proceedings of the world congress on engineering and computer science*, vol. 1. Citeseer, 2008.

[76] M. Pelevina, N. Arefyev, C. Biemann, and A. Panchenko, "Making sense of word embeddings," *arXiv preprint arXiv:1708.03390*, 2017.

[77] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162

[78] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[79] L. Portnoy, "Intrusion detection with unlabeled data using clustering," Ph.D. dissertation, Columbia University, 2000.

[80] C. M. Rosenberg and L. Moonen, "Spectrum-based log diagnosis," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3382494.3410684

[81] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[82] G. Seif. The 5 clustering algorithms data scientists need to know. [Online]. Available: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

[83] 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)*, 2013, pp. 402–411.

[84] A. Smiti and Z. Elouedi, "Dbscan-gm: An improved clustering method based on gaussian means and dbscan techniques," in *2012 IEEE 16th international*

conference on intelligent engineering systems (INES). IEEE, 2012, pp. 573–578.

[85] Z. Svirca. Density-based algorithms. [Online]. Available: https://towardsdatascience.com/density-based-algorithms-49237773c73b

[86] A. Toshniwal, K. Mahesh, and R. Jayashree, "Overview of anomaly detection techniques in machine learning," 2020, pp. 808–815, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85097830647&doi=10.1109%2fI-SMAC49090.2020.9243329&partnerID=40&md5=949d4175f286d972b170d8157ca22074

[87] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Long short-term memory based operation log anomaly detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 236–242.

[88] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[89] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[90] 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*, 2009, pp. 117–132.

[91] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Mining console logs for large-scale system problem detection." *SysML*, vol. 8, pp. 4–4, 2008.

[92] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 102–112.

[93] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3338906.3338931

Appendix A

Supplemental Information

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden