Anomaly Detection in Log Files Using Machine Learning Techniques

Lakshmi Geethanjali Mandagondi

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):
Lakshmi Geethanjali Mandagondi
E-mail: lama18@student.bth.se

University advisor:
Abbas Cheddad
Department of Computer Science

External advisor:
Patrik Olesen
Patrik.olesen@ericsson.com

External advisor:
Simon Bood
Simon.bood@ericsson.com

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

Context Log files are produced in most larger computer systems today which contain highly valuable information about the behavior of the system and thus they are consulted fairly often in order to analyze behavioral aspects of the system. Because of the very high number of log entries produced in some systems, it is however extremely difficult to seek out relevant information in these files. Computer-based log analysis techniques are therefore indispensable for the method of finding relevant data in log files.

Objectives The major problem is to find important events in log files. Events in the test suite such as connections error or disruption are not considered as abnormal events. Rather the events which cause system interruption must be considered as abnormal events. The goal is to use machine learning techniques to "learn" what an "expected" behavior of a particular test suite is. This means that the system must be able to learn to distinguish between a log file which has an anomaly, and which does not have an anomaly based on the previous sequences.

Methods Various algorithms are implemented and compared to other existing algorithms based on their performance. The algorithms are executed on a parsed set of labeled log files and are evaluated by analyzing the anomalous events contained in the log files by conducting an experiment using the algorithms. The algorithms used were Local Outlier Factor, Random Forest and Term Frequency Inverse Document Frequency. We then use clustering using KMeans and PCA to gain some valuable insights from the data by observing groups of data points to find the anomalous events.

Results The results show that Term Frequency Inverse Document Frequency method works better in finding the anomalous events in the data compared to other two approaches after conducting an experiment which are discussed in detail.

Conclusions The results will help developers to find the anomalous events without manually looking at the log file row by row. The model provides the events which are behaving differently compared to the rest of the event in the log and that cause the system to interrupt.

Keywords: Anomaly Detection, Log Files, Machine Learning, Clustering, Outlier Detection.
Acknowledgments

I am very grateful for the support given by my supervisors. Thank you, Associate Prof. Abbas Cheddad, for your time, feedback, and guidance throughout the project.

I would like to express my gratitude to my parents, sister (namely Chandra Sekhar, Janaki, and Anjali) who encouraged and helped me and stood as my support system throughout this journey.

Thank you Lucas Jönefors for giving me an opportunity to work with Ericsson. I am very thankful to my industrial supervisor Patrik Olesen for your involvement and continuous mentorship throughout the project and pushing me further than I thought I would go. I would also like to thank Simon Bood for your active participation in the discussions and for always being helpful. Sincere thanks to all colleagues who helped me throughout this project.

I am grateful to all the lecturers and professors at Blekinge Institute of Technology for their support and guidance towards the successful completion of my studies.

Finally, special thanks to all my family members and friends who believed in me.
Contents

Abstract i

Acknowledgments iii

1 Introduction 1
   1.1 Research Problem ............................. 1

2 Aim and Objectives 3
   2.1 Aim and Objectives ............................ 3
   2.2 Research Questions ............................ 3
   2.3 Outline ................................... 4

3 Background 5
   3.1 Machine Learning ............................. 5
   3.2 Supervised Learning ........................... 5
   3.3 Unsupervised Learning .......................... 5
   3.4 Clustering ................................. 5
   3.5 Logs .................................... 6
   3.6 Anomalies .................................. 6
   3.7 Drain .................................... 6
   3.8 Random Forest .............................. 7
   3.9 Local Outlier Factor ........................... 8
   3.10 Term Frequency Inverse Document Frequency (TFIDF) ........ 8
   3.11 K-Means .................................. 9
   3.12 Principal Component Analysis (PCA) .................. 9
   3.13 Accuracy .................................. 10
   3.14 Precision .................................. 10
   3.15 Recall ................................... 10
   3.16 F1 Score .................................. 10

4 Related Work 11
   4.1 Limitations from Related Work ..................... 13
   4.2 Contribution ................................ 13

5 Method 15
   5.1 Literature Review ................................ 15
      5.1.1 Snowballing(SB) ............................ 16
   5.2 Experiment .................................. 17
List of Figures

3.1 A typical test log .................................. 6
3.2 Random Forest Classifier .......................... 7

5.1 System Configuration ............................. 18
5.2 Workflow of the research .......................... 19
5.3 An Overview of Maoni ............................ 20
5.4 An Overview of JCAT ............................ 20
5.5 Given format for log files ......................... 21
5.6 Workflow of Drain Parser ......................... 22
5.7 Regex code used for parsing the log files ........ 22
5.8 A structured file .................................. 23
5.9 A template file .................................. 24
5.10 LOF score of the TemplateId values in log file when K=20 .... 25
5.11 A fully parsed file .............................. 26
5.12 Sliding window for the log data ................... 27
5.13 Parsed log file .................................. 28
5.14 Dataframe with log files having only occurrences taken as input ... 28
5.15 TFIDF values for logs w.r.t EventId ............... 29
5.16 TFIDF values for logs w.r.t EventId ............... 30
5.17 Formula to calculate cosine similarity .............. 30
5.18 Cosine Similarity values of the documents .......... 31
5.19 Elbow Chart .................................... 32

6.1 LOF performance .................................. 33
6.2 Plot of LinId and TemplateId when K=5 ............... 34
6.3 Plot of LinId and TemplateId when K=30 ............... 34
6.4 Performance of Random Forest ....................... 35
6.5 Plot of Actual and Predicted values on the data ....... 35
6.6 K-Means Plot for the log files .................... 36
6.7 Elbow Chart .................................... 37
6.8 Principal Component selection using PCA ............. 37
6.9 Heat map for the tfidf values of the log files ....... 38
6.10 Performance metrics for approach 3 ................ 39
6.11 PCA plot for logs 8952 and 8953 ................... 39
6.12 PCA plot for logs 8954 and 8955 ................... 39
6.13 PCA plot for logs 8956 and 8957 ................... 40
6.14 PCA plot for logs 8958 and 8959 ................... 40
6.15 PCA plot for logs 11900 and 11901 ................. 41
6.16 PCA plot for logs 11902 and 11903 ........................................ 41
6.17 PCA plot for logs 11904 and 11905 ........................................ 42
6.18 PCA plot for logs 11906 and 11907 ........................................ 42

7.1 Performance metrics .......................................................... 43
7.2 Events having different sequence pattern from the rest .......... 44
Nomenclature

\textit{JCAT}  Java Common Auto Tester

\textit{NLP}  Natural Language Processing

\textit{PCA}  Principal Component Analysis

\textit{SI}  Silhouette Index

\textit{SSE}  Sum of Squared Errors

\textit{TFIDF}  Term Frequency Inverse Document Frequency
Chapter 1

Introduction

Logging is a common programming practice of significant importance to gather system run-time information for postmortem analysis[47]. Various products are tested to make sure that the products are safe to use. This will create a log file by collecting the run-time information. Log information is all around accessible in almost all computer systems. In identifying the critical points for debugging a system failure and performing root cause analysis, the log has a vital role by recording system state and its significant events causing those critical points[36].

In understanding the system status and its performance issues the log data is considered to be an important and valuable resource. Therefore, for online monitoring of the system or detecting anomaly, the source of information is naturally present in the logs [42].

The advantage of log files is that they keep track of every single event that is carried out, including artifacts of attacks. As most of the logs are designed to be human-readable, they often contain text messages and also provide information about parameters and other values related to the currently running processes. There are uncountable different ways how log files are structured in practice and the contents of most real-world log files exhibit highly different features as they depend on the type of application, configurations defining what type of messages are logged (e.g., informative messages, errors or debug output), the verbosity of the log lines, what kind of components are placed in the system and in which way they are writing their messages to the log file [44].

1.1 Research Problem

At Ericsson, the hardware and software products are tested in automated tests. Some tests run continuously, and others run once a day or less often. The tests are grouped into test-suites depending on functionality and area. On average each suite takes about 2-4 hours to run. If one test fails, the result is logged but the execution of the suites continues. This produces a large log. The log is a raw text file and it contains information about each performed test step as well as errors from the test system and other printouts from the embedded software.

A tool called JCAT (Java Common Auto Tester) is used to create reports from these logs, the JCAT is a feature and system testing tool used by various Ericsson
organizations. The report shows each test step and the result of that particular test. Any errors or failed tests are highlighted and are easy to find but the reason for the failure is challenging to observe. Mainly, there are two types of log files; the test logs and the system logs.

The test logs contain printouts from several resources which include:

- Logs from the test framework, which prints all the test steps and the results.
- Logs from the Java test code that contains information, errors, and exceptions.
- Logs from various Java managers are used by the test code (for example: connection manager) which tells about the connection attempts.
- Finally, logs from the embedded software, in which some printouts from the system log are added to the test log.

Examining the test log manually is very time consuming and trying to find out the cause for the failure of the test is tedious. Detecting the root cause for the failure of the log will be much easier when we train a machine to detect the failure and it also consumes less time than the manual work time. The order of the logs is not trivial as the system is multi-threaded.

The system logs record all the errors, warnings, notifications and information that comes from the embedded software that runs on the products (or hardware). These logs are uniform i.e. they start with a timestamp and what subsystem they come from and so forth. They do not need any kind of pre-processing and we can understand the content of the log much better as compared to that of the test log. But the problem with the system log is that it is updated constantly. There is no beginning or ending and the log is a “circular file” so it will eventually be overwritten with new data. So, this makes it harder to get good training sets from these. Whereas, the advantage of the test log is that we can repeat the scenario that causes the log, either by choosing the regression suites from where the log file forms or by running our tests. We will get a log that looks similar every time we run the test (at least after pre-processing). This makes it easier to get good training sets. The logging will be finished before we examine the log file i.e. the log is not updated during the inspection. So, the training phase would not be affected by this issue. Based on the above, we choose test logs as input files for implementing the models as well as throughout the thesis.

Logs relating to disruption and lost connection are common but detecting such anomalies would not affect the result. Detecting the flaws that may cause an interruption in the process is important. The test step will produce the same log; every time if there are no changes or flaws. Configuration of a test can sometimes have unexpected side effects on the system.
2.1 Aim and Objectives

The main purpose of this project is to develop a program using machine learning techniques that can read the test log file and learn what a "normal" log should look like and from that it should be able to determine that a log is “abnormal”. But it should not only be the case for fault finding. The detected anomalies can be of intrigue regardless of whether all tests in the suite have passed so that it will assist with telling us what causes flaws. The test performs the configuration of hardware and checks if the configuration has had the expected result. However, sometimes tests fail because of some unexpected changes occurred. It can also be because of a previous test that has not cleaned up properly.

Objective 1 Implement an application that can parse a text file (the automated test log) and analyze its content to detect unexpected behaviors that cause system interruptions.

Objective 2 Use machine learning techniques to learn what an expected behavior of a particular test suite is. This means that the system must be able to learn to distinguish between a log file which has an anomaly, and which does not have an anomaly based on the previous sequences.

2.2 Research Questions

RQ1 Which deep/machine learning techniques can be used to develop a model that best predicts the anomalies in the test log files?

This was answered by conducting a literature review. After conducting the literature review various machine learning models that are suitable for this research are selected.

RQ2 How do these techniques perform on our addressed problem and can we improve the accuracy by recognizing anomalies from the test log files?

This was answered by conducting an experiment. The selected machine learning models are made to predict the anomalous events from the test logs and
they are compared against each other to know the model which gives best results.

2.3 Outline

The rest of the thesis is organized as follows.

- Chapter 2 discusses aim and objectives which gives a brief description of the goals that will be accomplished by the end of this thesis.

- Chapter 3 discusses the background which gives information of the machine learning and the models used in this thesis.

- Chapter 4 analyzes the related work that includes previous research works related to this thesis.

- The approach of how this thesis was performed is explained in Chapter 5 where experiment was performed to evaluate the effectiveness of the method implemented.

- In Chapter 6 experimental results are provided.

- Chapter 7 discusses the obtained results and compares the models performance.

- Chapter 8 tells about the challenges and limitations faced during the project and how these limitations are overcome.

- Chapter 9 discusses the conclusions drawn and the future work.
3.1 Machine Learning

Machine Learning is a form of artificial intelligence that gives computers the skills to learn without being specifically programmed. It focuses on building computer programs that are subject to change when exposed to new data. It can be classified as either supervised or unsupervised. It is about using the right features to build the right models that achieve the right tasks. These tasks include binary and multi-class classification, regression clustering and descriptive modeling.

3.2 Supervised Learning

Supervised learning requires the availability labelled data as algorithms falling under this umbrella can apply past knowledge to new data [40].

3.3 Unsupervised Learning

On the other hand, learning from unlabeled data is called unsupervised learning. For instance, to evaluate particular data into clusters, one can calculate the average distance from the cluster centers. Other forms of unsupervised learning include learning associations and identifying hidden variables such as film genres. Overfitting is a concern in supervised learning; for instance, assigning each data point its cluster will reduce the average distance to the cluster center to zero, yet is not very useful. These algorithms make conclusions from the datasets [40].

3.4 Clustering

The task of grouping data without prior information on the groups is called clustering. A typical clustering algorithm works by assessing the similarity between instances and putting similar instances in the same cluster and dissimilar instances in different clusters [37].
3.5 Logs

Large data system logs are typically unstructured data printed in time sequence. Normally, each log entry (line) can be divided into two different parts: constant and variable. The constant part is the messages printed directly by statements in source code. Log keys are often extracted from these constant parts, where log keys are the common constant messages altogether similar log entries. A typical test log looks like this:

![Figure 3.1: A typical test log](image)

In the above figure, the log starts with a timestamp followed by a type event INFO or DEBUG. This indicates the type of message the event generates in the log file. After the event, a description of the event is generated followed by the commands and the test steps performed while running the tests.

3.6 Anomalies

An anomaly is an abnormality that does not fit with the rest of the pattern. The word anomaly comes from the Greek word "anomolia" meaning uneven or irregular. When something is unusual compared to the things around it, it is called an anomaly.

3.7 Drain

Drain is an online log parser that can parse a large volume of logs in a streaming and timely manner. It uses a fixed depth parse tree that encodes specially designed parsing rules. Most of the existing log parsing methods focus on offline, batch processing of logs. As the volume of logs increases rapidly, model training of offline log parsing methods becomes time-consuming. Drain uses a fixed depth tree to guide the log group search process which effectively avoids constructing a very deep and unbalanced tree.

The goal of this log parsing is to transform raw log messages into structured log messages. When a new raw log message arrives Drain will preprocess it by simple regular expressions supported domain knowledge. Then a log group is searched (i.e., leaf node of the tree) by following the specially designed rules encoded in the internal
3.8 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees’ habit of over-fitting to their training set. As a part of their construction, random forest predictors naturally cause a dissimilarity measure among the observations. One also can define a random forest dissimilarity measure between unlabeled data: the thought is to construct a random forest predictor that distinguishes the “observed” data from suitably generated synthetic data. The observed data are the original unlabeled data and the synthetic data are drawn from a reference distribution. Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way.

For data including categorical variables with a different number of levels, random forests are biased in favor of these attributes with more levels. Methods such as partial permutations and growing unbiased trees can be used to solve the problem. If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups [30]. Fig 3.2 shows the working of a random forest classifier [6]:

![Random Forest Classifier](image)

Figure 3.2: Random Forest Classifier

From the Fig 3.2, labeled data is divided into training and validation. Then the
training data is further divided into subsets of data where each subset is a decision tree. The random forest classifier is a multitude of decision tree, where each tree predicts the output the predictions of all trees are combined and averaged to give the best possible prediction. Validation set is used to predict the values against the training data which determines the performance of the model.

3.9 Local Outlier Factor

The local outlier factor is predicted on an idea of a local density, where locality is given by k nearest neighbors, whose distance is employed to estimate the density. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of comparable density, and points that have a substantially lower density than their neighbors. These are considered to be outliers. The local density is estimated by the standard distance at which some extent are often "reached" from its neighbors. The definition of "reachability distance" utilized in LOF is a further measure to supply more stable results within clusters.

Let k-distance(A) be the distance of an object A to the k-th nearest neighbor. Note that the set of the k nearest neighbors includes all objects at this distance, which may within the case of a "tie" be quite k objects. We denote the set of k nearest neighbors as Nk(A). A value of approximately 1 indicates that the object is comparable to its neighbors (and thus not an outlier). A value below 1 indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outliers [13].

- LOF(k) 1 means Similar density as neighbors
- LOF(k) < 1 means Higher density than neighbors (Inlier)
- LOF(k) > 1 means Lower density than neighbors (Outlier)

3.10 Term Frequency Inverse Document Frequency (TFIDF)

TFIDF is the term weighting method that has been continuously applied to assign term weights in support of text mining, document modeling, text categorization, text clustering, and text summarization. This approach involves the use of term frequency and inverse document frequency components. TF refers to the frequency at which a term is found in a particular document, and IDF refers to how frequent a term is found in all documents examined. The TFIDF value of a corresponding term must therefore be higher when this term occurs frequently in a particular document, but it must also occur rarely in the whole set of documents examined [34].

Jones K S proposed the IDF idea firstly in 1972, He pointed out that: in a set of documents, if the higher feature items appear in all the document, less information entropy it contains, the corresponding weight should be lower; if a certain feature
3.11. K-Means

K-means clustering may be a method of vector quantization, originally from signal processing, that aims to partition $n$ observations into $k$ clusters during which each observation belongs to the cluster with the closest mean, serving as a prototype of the cluster. This algorithm clusters data by trying to separate samples in $n$ groups of equal variances, minimizing a criterion referred to as the inertia or within-cluster sum-of-squares. It requires the number of clusters to be specified. This algorithm is divided into a set of $N$ samples $X$ into $K$ disjoint clusters $C$, each described by the mean of the samples in the cluster. The means are commonly called as the cluster “centroids”. This algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion. Inertia is the measure of how internally coherent clusters are.

K-means is often referred to as Lloyd’s algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the foremost basic method being to steal on $k$ samples from the dataset $X$. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean of all of the samples assigned to each previous centroid. The difference between the old and therefore the new centroids are computed, and the algorithm repeats these last two steps until this value is a smaller amount than a threshold. In other words, it repeats until the centroids do not move significantly[33].

3.12 Principal Component Analysis (PCA)

PCA is a statistical method that captures patterns in high-dimensional data by automatically choosing a set of coordinates the principal components that reflect covariation among the original coordinates. We use PCA to separate out repeating events in feature vectors, thereby making abnormal message patterns easier to detect. PCA has run time linear in the number of feature vectors, so the anomaly detection can scale to large log files [35]. It is one of the most commonly used techniques in multivariate analysis for dimension reduction and feature extraction, and...
is particularly well suited for where the data is high-dimensional. PCA has a wide range of applications ranging from data compression to clustering.

The metrics used to evaluate the performance of the approaches used are mentioned below:

### 3.13 Accuracy

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. If we have high accuracy for a particular model then the model is best [4].

\[
Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.1}
\]

### 3.14 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate[4].

\[
Precision = \frac{TP}{TP + FP} \tag{3.2}
\]

### 3.15 Recall

Recall is the ratio of correctly predicted positive observations to the all observations in actual class[4]. It has values between 0 and 1. Higher the recall, better performing the model is.

\[
Recall = \frac{TP}{TP + FN} \tag{3.3}
\]

### 3.16 F1 Score

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account[4]. It ranges from 0 and 1, higher the F1 score better the model is performing.

\[
F1Score = 2 \ast (Recall \ast Precision)/(Recall + Precision) \tag{3.4}
\]
Chapter 4

Related Work

Ericsson provides data and tools for testing which will be utilized for the log analysis. Besides that, for writing the report, performing a literature review will help in better understanding the work done earlier on the methods to overcome similar issues. This will help us to identify techniques that will achieve the aim and objectives of this research.

In the paper[42], cross-predict logging prediction methods and techniques were investigated. They proposed a method called EC-Logger which is a novel cross-project ensemble-based catch-block logging prediction model. Nine base classifiers were used and combined using ensemble techniques. The performance of EC-Logger was evaluated on three open-source Java projects: Tomcat, Cloud Stack, and Hadoop. The classifier is based on ensemble techniques, like bagging, average vote, and majority vote outperforms the baseline classifier. Overall, the EC-Logger Average Vote model performs best. The results show that the Cloud Stack project is more generalizable than the other projects. This paper relates to my research problem as we try different machine learning algorithms and pick the best performing one. They combined the techniques using the ensemble method to choose the best performing model.

In the paper[43], they proposed the LogOpt tool for automated catch block logging prediction. LogOpt is based on a machine learning framework and uses static features from source code to train the model. They identified 46 distinguishing features from source code for logging prediction. They presented results of the evaluation of LogOpt on two large open-source projects (Apache Tomcat and CloudStack). LogOpt was found to be effective in catch block logging prediction and gave an f1 score of 93% with 88.26% precision and 99.02% recall on the CloudStack project when used in combination with random forest classifier. Whereas, we try to distinguish features (log patterns) from the code for predicting the anomalies in the log by choosing the best performing machine learning algorithm from the selected algorithms.

In the paper[36], they proposed DeepLog, a deep neural network model utilizing Long Short-Term Memory (LSTM), to model a system log as a natural language sequence. This allows DeepLog to automatically learn log patterns from normal execution and detect anomalies when log patterns deviate from the model trained from log data under normal execution. Similarly, we build a system to automatically learn the log patterns from a normal log file and detect the anomalies when the log patterns exhibit a different pattern, but this thesis work will try to deploy and explore the
performance of other machine learning techniques (see the Method section for more details), which would tease it apart from prior related works [36][42]. At Ericsson, log data is generated every hour and tests run continuously.

In paper [44] an online anomaly detection approach that displays security relevant metrics as time-series and employs forecasting models in order to detect deviations from expected behavior. A clustering model that is able to connect log line clusters from a sequence of static cluster maps and thereby supports the detection of transitions between the clusters. The main feature of their approach is contextual anomalies i.e., log lines that do no cohere to previously gained knowledge about their average frequency of occurrence, periodicity and correlation are detected. This made them to detect highly dissimilar lines which occur only once as outliers rather than temporal anomalies which are observed as system behavioural changes over time. This approach is self-learning does not require any previous knowledge about attacks or the structure and the content of log data.

Paper [29] viewed text categorization as a process of category search. The documents are partitioned into clusters and are compared with each cluster rather than each document. Cluster based searches have been used to improve both the efficiency and effectiveness of the full search. They have compared four category search strategies.

In [35], it aims to detect frequent patterns from the log files to build normal profiles and then identify anomalous behaviour from the log files. They designed a fast and efficient algorithm to detect line patterns from raw log files. The algorithm relied on the nature of log files. Their choice is the employment of data clustering algorithm. They take the whole log, and record every word, position and its occurrence times and build the cluster candidates table based on the frequent words that get at the first step. When a line is found to have more than one frequent word, its a cluster candidate. The last step of the algorithm is to generate clusters from candidate table, all candidates with count value are greater than the threshold value are taken as the cluster. They have investigated a variety of such methods and have found that Principal Component Analysis (PCA) combined with term-weighting techniques from information retrieval yields excellent anomaly detection results on both feature matrices, while requiring little parameter tuning. As with frequent pattern mining, the goal of PCA is to discover the statistically dominant patterns and thereby identify anomalies inside data.

In paper [41], they used k-means model for separating anomalous and normal events in highly coherent clusters. XGBoost was implemented as a gradient tree boosting algorithm, that used the previous binary clustered data for producing a set of simple interpretable rules. The rules represented the rationale for generalizing its application over a massive number of unseen events in a distributed computing environment. Based on this they obtained classified anomaly events.

In paper [45], they proposed a hybrid technique that combines data mining approaches like K Means clustering algorithm and RBF kernel function of Support Vector Machine as a classification module. The main purpose of their technique is to
4.1 Limitations from Related Work

- From paper [36], we similarly build a system to automatically learn the log patterns from a normal log file and detect the anomalies when the log patterns exhibit a different pattern, but this thesis work will try to deploy and explore the performance of other machine learning techniques (see the Method section for more details), which would tease it apart from prior related works [36][42]. At Ericsson, log data is generated every hour and tests run continuously.

- In paper [44], they used a clustering model to detect contextual anomalies whereas in this thesis I have used three various algorithms to find the best suitable one which predicts anomalies with good efficiency.

- In [29], they have partitioned the documents into clusters and compared with each cluster. Whereas, in this thesis various algorithms have been implemented and clustering was done to compare the documents against each other.

- Finally, in paper [35] they used pattern mining and clustering to detect the frequent patterns from log files to identify anomalous behaviour from raw log files. In this thesis, raw logs were parsed first and then various algorithms have been implemented on the parsed data.

4.2 Contribution

Since, this research was done in pair at Ericsson, I have mentioned only my contribution of work in this thesis report.

Despite these encouraging results, not many studies have performed research using Random Forest, Local Outlier Factor and the combination of TFIDF, K-Means and PCA. In this study, I proposed a new combination TFIDF+KMeans+PCA inspired from the study [41] [45] which produce best results for this research. The data set used in this research is internal Ericsson data which needs to be analyzed and pre-processed carefully so that there wouldn’t be any loss of useful data needed for this research. My contribution to working in this area is analyzing the test logs to find a suitable parsing technique to clean them and detecting the problem that occurred in the test logs by using machine learning techniques and finding a suitable method to solve it.
In this research, two methods are used to answer the research questions: Literature Review and Experiment. The design of this project is to analyze the data and consider suitable methods that can be used to find the anomalies from the log files. So, a literature review is conducted to gain knowledge on the machine learning techniques used previously for this sort of issues, we have also examined other deep learning architectures in other domains that could potentially address the problem at hand. This helped us in selecting the algorithms that can be used to develop the model. The selection of algorithms depends on how accurately they will predict the anomalies, which means selecting algorithms that show the best performance results by using conventional statistical measurement metrics. A literature review (survey) of existing machine learning algorithms in the domain that could potentially work best for this problem (i.e., finding the anomalies in the log file) gives an answer to RQ1.

RQ2 is answered by the experiment conducted. Experiment is chosen over other research methods as it involves manipulation of variables [46] that is necessary which is important to obtain the results in this study. The type of problem addressed is dealt by classification as the algorithm needs to classify an anomaly from the log file. The data which is collected is in its raw form and not fully structured or labeled. So, the log file is labeled while parsing the data, later only the data required to feed the algorithms was taken into consideration. This makes it labeled data which is why we used supervised machine learning techniques in the study. Whereas, unsupervised techniques are used when necessary which is discussed in the later sections.

5.1 Literature Review

A review of the literature is a compilation of research that has been published on a topic by recognized scholars and researcher[5]. There are two main types of conducting these type of searches. They are snow balling and database searches. Data search is performed in ACM, IEEE Explorer, Science Direct using the following search strings:

- ("Anomaly detection in Log Files" OR "Time series forecasting using machine learning" OR "Anomaly detection and diagnosis from system logs" OR "Cluster Analysis for log data").
5.1.1 Snowballing (SB)

In this search, the reference list and citations of relevant papers are reviewed to identify new papers. This means, we initially take the set of papers after database search thereafter conducted forward and backward snowballing on this set [31].

Backward snowballing (BSB)

The sampling done using the reference list of the papers in the start set is named as BSB. The following are reviewed in a BSB [31]:

- Title of the referenced paper.
- The point of reference of the paper.
- Abstract of the referenced paper.

Forward Snowballing (FSB)

The sampling done by reviewing the citations is termed as FSB. The citations were retrieved from Google Scholar. The papers that were included in the iterations were added to the start set and SB of the newly added papers was done in the next iterations. This process was followed until no new papers were found. When all the papers were added to the start set it was considered as the final set, which included all primary studies. In FBS the following order is followed [31]:

- Title of the paper citing.
- Abstract of the paper citing.
- The point of reference to the paper being cited.

Papers were selected based on the following criteria:

Inclusion Criteria

- Papers based on anomaly detection which are written in English.
- Papers based on clustering techniques
- Papers which are in English
- Papers focusing on the research problem.

Exclusion Criteria

- Papers that were not published.
- Papers that does not address the research problem and research questions.
- Papers that does not focus on the search string.
During this search, previous works have been studied to find what methods have been performed to solve similar problems. The first set includes [36][47][21] and forward and backward snowballing has been performed on these. This helped us in getting an insight on the what techniques can be implemented for this project. Table 5.1 shows papers excluded at various steps in this process. The machine learning techniques which we decide to use, must be able to determine what is useful and what is not useful information in the test log. This was done under the guidance of my supervisor in Ericsson who is a professional in this field.

The selected algorithms were Local Outlier Factor, Random Forest and combination of Term Frequency Inverse Document Frequency, K-Means, PCA. The motivation behind choosing these algorithms is mentioned in Chapter 5.

## 5.2 Experiment

This section is mainly divided into three parts: Experiment setup, data preprocessing, and implementation of the approaches chosen. In the Experiment set up, we know the details in which environment the experiment is conducted. In data preprocessing, an explanation of the parsing using Drain is done in detail. In the approaches section, the methods that were chosen after conducting the literature review are implemented.

### 5.2.1 Experiment Setup

**Environment**

The specifications of the system are mentioned in the Figure 5.1. The programming languages used in the research was Python. Various python and machine learning libraries were used to which were needed for the implementation of the algorithms. The reason behind choosing python is that it is one of the most accessible programming languages available as it has more simplified syntax which gives more emphasis on natural language [27]. Python for developing complex scientific and numeric applications. Python is designed with features to facilitate data analysis and visualization the data visualization libraries and APIs provided by Python help you to visualize and present data in a more appealing and effective way [19].
Libraries Used

**Scikit:** Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy [22].

**Scipy:** SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering [23].

**NumPy:** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays [16].

**Pandas:** pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series [18].

**Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy [14].

**Seaborn:** Seaborn provides an API on top of Matplotlib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas [14].

System logs perform a critical function in software-intensive systems as logs record the state of the system and significant events in the system at important points in time. Unfortunately, log entries are typically created in an ad-hoc, unstructured, and uncoordinated fashion, limiting their usefulness for analytics and machine learning [32]. Additionally, the text file provided by Ericsson is a also raw log file (test log). So, we pre-processed the data and only the data will be useful is assembled. Then the assembled data is observed to detect where the anomaly has occurred. For the machine to detect a normal or an abnormal file, the data is trained, and the anomalies of various types are grouped accordingly.
In this research, the independent variable is software environment and the dependent variable is log file. Since the software environment in which the test were run is not affected by any other variables involved in this study that becomes an independent variable. While, the logs are generated after the tests run, they change if the test is conducted in a different environment.

After the literature review, an experiment is conducted by testing the chosen algorithms and comparing their performances by using the statistical metrics. The drawbacks mentioned in the RQ2 refer to events (printouts) that could be an explanation for a failed test. Since the tests perform configuration of the products i.e. real hardware, a printout saying something like “Waiting for the signal to lock timed out” is probably more relevant than a printout saying “did not get a connection to xxx.xxx.xxx.xxx. trying again”. It is considered abnormal even if two of the above cases occur. So, these kinds of messages in the log file will be considered as a flaw.

5.2.2 Implementation

Data Collection

The data is taken from the Maoni, a flexible tool for visualizing the autotest outcomes from MINI-LINK’s CI Machinery (used in Ericsson). Its primary purpose is to visualize software regressions, to help understand quality levels, and sources of intermittent test results. Maoni’s primary view is the Matrix view which represents the user with a powerful filterable view into the results of executed automated tests. The matrix view contains a filter bar on top where multiple filters can be applied to reduce the amount of data visualized in the matrix. The target time period (“Test Date”) can be changed, which shows the tests run on that particular date.

The status of particular execution is also represented, i.e., passed/failed/skipped/excluded. The Maoni has all the log files, which represent the successful and failed test cases where the test cases are run continuously. The green ones represent successful test cases whereas the red represents failed ones. When you click on a particular test
Chapter 5. Method

Figure 5.3: An Overview of Maoni

Maoni suite opens in a new window where the log data is seen in JCAT (Java Common Auto Tester). It is a Java-based test automation framework for Ericsson products and is an inner-source tool (i.e. open-source within Ericsson). JCAT is used to make a readable report out of the text logs that are produced from the embedded software. Logs are downloaded from here and then pre-processing of the data is done. A typical log from JCAT looks like this:

Figure 5.4: An Overview of JCAT

5.2.3 Data Pre-processing

Parsing

After collecting the data from Maoni, we parse the data using Drain. We have many parsing techniques, but the parser used in this experiment is Drain, which is an online log parser. The test log files are parsed, which made it easier to implement various
algorithms on the data. We implement supervised machine learning techniques on the parsed log file where the algorithms are trained to investigate a couple of parsed sequences, to be able to find the anomalies in the non-parsed log files based on these sequences. The methods will be compared based on the following common performance metrics: classification accuracy, cosine similarity, silhouette index.

Drain is a fixed depth tree based on online log parsing method which was suggested by the supervisor at Ericsson. We are using a function ‘CleanUntilStart’ to select the part of the log that we are going to parse. We are taking a part of a log event from Test Step [47] to Test Step [15]. The goal of this parsing is to transform raw log messages into structured log messages. Raw log messages are unstructured data which includes timestamps and raw message contents.

In the parsing process, the parser distinguishes between a constant part and a variable part of each raw log message. The constant part is tokens that describe a system operation template (i.e., log event) while the variable part is the remaining tokens that carry dynamic run-time system information[39].

Each log group has two parts: log event and log IDs. Log event is the template that best describes the log messages during this group that consists of the constant part of a log message which is useful for this research. Log IDs records the IDs of log messages in this group. Log format is mentioned while parsing to understand the fields in the test log clearly as shown in Figure 5.5. One special design of the parse tree is that the depth of all leaf nodes is the same and are fixed by a predefined parameter depth. This parameter bounds the number of nodes Drain visits during the search process, which greatly improves its efficiency [39].

\[
\text{log\_format} = \langle \text{Time} \rangle, \langle \text{Seconds} \rangle, \langle \text{Level} \rangle, \langle \text{Component} \rangle, \langle \text{Level2} \rangle, \langle \text{Content} \rangle
\]

Figure 5.5: Given format for log files

**Step 1: Pre-process by domain knowledge**

The drain allows users to give simple regular expressions supported on domain knowledge that represent commonly used variables, like IP address and block ID. Then Drain will remove the tokens matched from the raw log message by these regular expressions. For example, IP addresses are removed by \( r'(/|) ([0-9]+[\d])' \). The regular expressions employed during this step are often very simple because they are used to match tokens instead of log messages. Besides, a dataset usually requires only a couple of such regular expressions[39]. For our dataset, we had to write eight such regular expressions (shown in Figure 5.7) to remove all the unnecessary tokens from our data. Regular expressions are written in a block of regex code. Expressions for block id, IP, Port, and MAC addresses, numbers, null values, false and quoted words with or without spaces. The similarity threshold is given 0.1 and the depth of the tree is 10.

**Step 2: Search by Log Message Length**
Chapter 5. Method

Figure 5.6: Workflow of Drain Parser

Drain starts from the root node of the parse tree with the pre-processed log message. The 1-st layer nodes within the parse tree represent log groups whose log messages are of various log message lengths. By log message length, we mean the number of tokens in a log message. Here, Drain selects a path to a 1-st layer node
based on the log message length of the pre-processed log message. This is based on the idea that log messages with an equivalent log event will probably have the same log message length. Although log messages with the same log event may have different log message lengths, more often handled by simple post-processing [39].

**Step 3: Search by Preceding Tokens**

Drain traverses from a 1-st layer node, which is searched in step 2, to a leaf node. This step is based on the assumption that tokens in the beginning positions of a log message are more likely to be constants. Specifically, Drain selects the next internal node by the tokens in the beginning positions of the log message. For example, for the log message “Receive from node 4”, Drain traverses from 1-st layer node “Length: 4” to 2-nd layer node “Receive” because the token in the first position of the log message is “Receive”. Then Drain will traverse to the leaf node linked with internal node “Receive” and attend step 4. The number of internal nodes that Drain traverses during this step is (depth-2), where depth is that the parse tree parameter restricting the depth of all leaf nodes. Thus, there are (depth-2) layers that encode the first (depth-2) tokens in the log messages as search rules. In practice, Drain can consider more preceding tokens with larger depth settings. Note that if the depth is 2, Drain only considers the first layer used by step 2 [39].

After parsing the Drain generates two files form the log file: structured and template files. Two directories namely the input directory where the logs that need to be parsed are stored and the output directory is given where the structured and template files need to be located. The structured file displays LineID, Event Template, and the rest of the messages and the template file displays EventID and Occurrences. Instead of the structured files, we take only the template files as input to the models. The reason behind this is, we need only the EventID and the occurrences that will have the information related to the type of events occurred in the test logs which helps us gain information about the anomalies.

![Figure 5.8: A structured file](image)

After parsing, the data is assembled in a data frame. Most of the fields in the log files are strings (messages and other information). These fields do not provide much information regarding the anomalies and therefore are not given as an input to the model.

**LineId:** LineId represents the number of each row in the log file.
Chapter 5. Method

Figure 5.9: A template file

**Time:** The time at which a particular event occurs is displayed in this field.

**EventId:** EventId is of type string which represents the type of event that happened in a particular row of the log file.

There are 131 types of events that occur in the log files in total. Since these events are represented in strings these are given numerical values. These are given stored under the TemplateId column. This is now mapped to the LineId. The columns ‘LineId’ and ‘TemplateId’ in the log files are displayed. Since Line Id is the same for all the logs there is only one column for all the logs. Since these two are numerical values this is given as input to the models. For all the three approaches Template file is taken as input. Based on the LineId the Template Id of a log is predicted. If the sequence changes in a particular row then it is basically an anomaly, because in such cases exceptional events occur.

### 5.3 Approaches

#### 5.3.1 Local Outlier Factor

The local outlier factor is predicted on an idea of a local density, where locality is given by k nearest neighbors, whose distance is employed to estimate the density. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of comparable density, and points that have a substantially lower density than their neighbors. These points are considered as outliers [13]. These outliers appear to be meaningful and cannot be identified using the simple nearest neighbour approach [12], the reason behind choosing this algorithm. The parsed template file is given as input to the local outlier factor. The required libraries are imported from sklearn. The anomaly score of every sample is named as Local Outlier Factor. This will measure the local deviation of the density of a given file with reference to its neighbors. It is local therein the anomaly score depends on how isolated the object is with reference to the encompassing neighborhood. More precisely, the locality is given by k-nearest neighbors, whose distance is employed to estimate the local density. If the LOF is higher for observation, the more anomalous the observation. Outliers tend to have a large LOF score. Inliers have a value close to 1. [24]. The definition used to calculate LOF value:
5.3. Approaches

- Reachability distance. Reachability distance is equal to the maximum of \( \text{dist}(p, o) \) and k-distance(o):

\[
\text{reach} - \text{dist}_k(p, o) = \max\{k - \text{distance}(o), \text{dist}(p, o)\}
\]  

where \( k \) is a positive integer number and \( \text{dist}(p, o) \) is a distance between object \( p \) and \( o \). k-distance(o) is defined as a distance between object \( o \) and k-nearest neighbours [11].

![Figure 5.10: LOF score of the TemplateId values in log file when K=20](image)

The parameter \( k \), which represents the number of neighbors LOF calculation will be considering. This is used to compare the density of one point to the other points. The value of \( k \) must be carefully considered as small \( k \) looks only at nearby points and there is a great chance of missing noisy data. If the \( k \) is large, we will miss the local outliers [10]. From Fig 5.10 we will know the outlier points detected using LOF. The points with having -1 value indicates outliers and the points with value 1 indicates inliers.

5.3.2 Random Forest Classifier

As a part of their construction, random forest predictors naturally cause a dissimilarity measure among the observations. One also can define a random forest dissimilarity measure between unlabeled data: the thought is to construct a random forest predictor that distinguishes the “observed” data from suitably generated synthetic data [30]. This is the reason behind choosing this machine learning technique. From Figure 5.11, 8952,8953...9910 represents the test logs that are taken as input after parsing. The two fields namely LineId and TemplateId are considered. Log 89583 is the test log formed after a particular test failed. The rest of the logs are good logs. These are all performed on the same test case at various time intervals. These logs combined as dataset are given as input to the model. This helps to analyze whether
the model can predict the anomaly or not. The model predicts the sequence of the log files compared to one another. If in a particular row the sequence skips or another sequence appears it will be considered as an anomaly.

We use the sliding window approach to analyze a statistic over a finite duration of the data. In this method, a window of specified length moves over the data, sample by sample, and the statistic is computed over the data in the window. In the subsequent time steps to fill the window, the algorithm uses samples from the previous data frame. The moving statistic algorithms have a state and remember the previous data. The window is of finite length, making the algorithm a finite impulse response filter. The window length defines the length of the data over which the algorithm computes the statistic. The window moves as the new data comes in. If the window is large, the statistic computed is closer to the stationary statistic of the data. For data that does not change rapidly, use a long window to get a smoother statistic [25]. For data that changes fast, use a smaller window. For our data we took checked window sizes for 10, 20, etc. . . and the model performed better if the window size is 45.

The libraries required to implement random forest are imported from the sklearn. This data is split into training and testing. We import a library named train_test_split from sklearn to split the data. The log files from 8952-9910 in the data frame are taken to learn the sequence and it is predicted on the log 89583. The data is fit into the model and then predicted on the test data. This is a prediction of how well the system can predict new sequences in the log file. This is measured by accuracy. The correct predictions are if a sequence is true the model predicts it as true, if the sequence is false, the models predict that correctly as negative. The rest of the predictions are wrongly made by the model. This indicates how well the system can predict and performance of the model. It is measured by the confusion matrix, which displays the number of true positives, true negatives, false positives, false negatives. If the number of false positives is very large it means that the system is not able to predict correctly. If the number of true positives is very high, it means that the

<table>
<thead>
<tr>
<th>Lineld</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
<th>Templateid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7.0</td>
<td>7.0</td>
<td>7.0</td>
<td>7.0</td>
<td>7.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8.0</td>
<td>8.0</td>
<td>8.0</td>
<td>8.0</td>
<td>8.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9.0</td>
<td>9.0</td>
<td>9.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10.0</td>
<td>9.0</td>
<td>9.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11.0</td>
<td>6.0</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.11: A fully parsed file
5.3. Approaches

TFIDF+KMeans+PCA

It stands for term frequency and inverse document frequency. And the tfidf is a weight often used in text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tfidf weighting scheme are often used by search engines as a central tool in scoring and ranking a document’s relevance given a user query. Tfidf can be successfully used for stop-words filtering in various subject fields including text summarization and classification. This is the reason behind choosing this technique.

Typically, the tfidf weight is composed by two terms: the first computes the normalized Term Frequency (TF), the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears [26].

TF: Term Frequency

It measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more time in long documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of...
normalization:
\[ \text{TF}(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}} \] [26].

**IDF: Inverse Document Frequency**

which measures how important a term is. While computing TF, all events are considered equally important. However, it is known that certain events may appear a lot of times but have little importance. Thus we need to weigh down the frequent events while scaling up the rare ones, by computing the following:
\[ \text{IDF}(t) = \log_e \left( \frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \right) \] [26].

To use the textual data for predictive modeling, the text must be parsed to remove certain words—this process is called tokenization. These words need to be encoded as integers, or floating-point values for use as inputs in machine learning. This process is called feature extraction (or vectorization).

<table>
<thead>
<tr>
<th>EventId</th>
<th>EventTemplate</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 00613a2b</td>
<td>COLORDROP PCP &lt;-, Sent bits: &lt;-, Received bit...</td>
<td>2</td>
</tr>
<tr>
<td>1 00736eca</td>
<td>sendCommand wrote: {&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;,...}</td>
<td>0</td>
</tr>
<tr>
<td>2 006f0a42</td>
<td>tnA &quot;debug-logs&quot;: &lt;a &lt;+&gt;</td>
<td>0</td>
</tr>
<tr>
<td>3 00a02dd9</td>
<td>log class hsi_bfd_packet_class</td>
<td>0</td>
</tr>
<tr>
<td>4 00beb14a</td>
<td>sendCommand wrote: {&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,&lt;&gt;,...}</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.13: Parsed log file

![Figure 5.14: Dataframe with log files having only occurrences taken as input](image)
5.3. Approaches

**Count Vectorizer**: After the pre-processing of data, we now covert the data into a vector form using the count vectorizer. It converts a collection of documents to a matrix(vector) of token counts. It also enables the pre-processing of data before generating the vector representation. This functionality makes it a highly flexible feature representation module for text. This produces a sparse representation of counts using scipy.sparse.csr_matrix. As most documents will typically use a really small subset of the words used in the corpus, the resulting matrix will have many feature values that are zeros (typically more than 99 of them). To be able to store such a matrix in memory but also to speed up algebraic operations matrix/vector, implementations will typically use a sparse representation such as the implementations available in the Scipy.sparse package [1].

```
Created in 0.0009500980377197266 seconds

exec-ml66_hourly_npu1002_ml66_hourly_dualsmall_8952  \
00613a2b  0.00316
00736eca  0.00000
008f0a42  0.00000
00a02dd9  0.00000
00beb14a  0.00000
...  
ff52c40c  0.00000
ff8ee984  0.00000
ff989f15  0.00000
ffce2e71  0.00000
ffcd2637  0.00000

exec-ml66_hourly_npu1002_ml66_hourly_dualsmall_8953  \
00613a2b  0.003156
00736eca  0.00000
008f0a42  0.00000
00a02dd9  0.00000
00beb14a  0.00000
```

Figure 5.15: TFIDF values for logs w.r.t EventId
### Cosine Similarity

Cosine Similarity is a metric used to measure the similarity between the documents which are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors that are projected in a multi-dimensional space. Here, vectors mean the arrays that contain the word counts of the two documents. The cosine similarity is of great advantage because even if the two similar documents are far apart by the Euclidean distance because of the size, they could still have a smaller angle between them. Smaller the angle, the higher the similarity [2]. It is calculated by the below formula:

\[
similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}
\]

![Formula to calculate cosine similarity](image)

The below figure shows the cosine similarity of the log files with respect to their Tfidf values. If the value is near to 1 or 1 it means that the documents are very similar and if their value is near to 0 they are not similar.
5.3. Approaches

Figure 5.18: Cosine Similarity values of the documents

| [1. | 0.99994186 | 0.9910021 | 0.9914021 | 0.99142185 | 0.99144859 |
| 0.99142185 | 0.99144262 | 0.0040397 | 0.00399554 | 0.00404338 | 0.00409002 |
| 0.00408999 | 0.00403878 | 0.00407957 | 0.00408568 | 0.00408664 | 0.00407846 |
| 0.99594186 | 1. | 0.99083597 | 0.99083597 | 0.99083418 | 0.99083474 |
| 0.99083418 | 0.99085494 | 0.00404796 | 0.00405711 | 0.00405164 | 0.00409823 |
| 0.00409823 | 0.00404689 | 0.00408777 | 0.00409501 | 0.00409466 | 0.00408665 |
| [0.99140921 | 0.99083597 | 1. | 1. | 0.99989501 | 0.99989091 |
| 0.99989091 | 0.99998317 | 0.00405782 | 0.00401346 | 0.00406151 | 0.0040821 |
| 0.0040821 | 0.00405675 | 0.00409772 | 0.00410498 | 0.00410463 | 0.00409661 |
| 0.99140921 | 0.99083597 | 1. | 1. | 0.99995061 | 0.99989091 |
| 0.99995061 | 0.99998317 | 0.00405782 | 0.00401346 | 0.00406151 | 0.0040821 |
| 0.0040821 | 0.00405675 | 0.00409772 | 0.00410498 | 0.00410463 | 0.00409661 |
| 0.99142185 | 0.99083418 | 0.99995061 | 0.99995061 | 1. | 0.99995729 |
| 0.99995729 | 0.99997904 | 0.00406153 | 0.00401714 | 0.00406523 | 0.00411198 |
| 0.00411198 | 0.00405046 | 0.00410447 | 0.00410474 | 0.00410389 | 0.0041036 |
| 0.99144859 | 0.99084374 | 0.9998091 | 0.9998091 | 0.99995729 | 1. |
| 0.99995729 | 0.9998991 | 0.00407136 | 0.00402686 | 0.00407587 | 0.004122 |
| 0.00412197 | 0.00407036 | 0.00411147 | 0.00411875 | 0.00411841 | 0.0041035 |

**Clustering**

**K-Means**: K-means clustering is an unsupervised machine learning technique which is used to identify clusters of data objects in a dataset. There are many types of clustering methods but the k-means is one of the oldest and most approachable. The above traits make implementing k-means clustering reasonably straightforward to analyze the data. This is the reason behind preferring this method [20]. K-Means is not individually used for anomaly detection rather it is purely used to get insight on the data based on the clustering results which are used for PCA. This clustering results are then used to perform the metrics such as silhouette score analysis. To determine the optimal number of clusters in K-means we use an elbow chart.

**Elbow Chart**: The idea of this method is to run K-Means clustering on the dataset for a range of values k, and for each value of k calculate the sum of squared errors(SSE). Then plot a line chart of the SSE for each value of k. If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best [28].

**Silhouette score**: This score or silhouette coefficient is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1[3].

- **1**: Means clusters are well apart from each other and clearly distinguished.
- **0**: Means clusters are indifferent or the distance between the clusters is not significant.
- **-1**: Means the clusters are assigned in the wrong way.

Silhouette score= \( \frac{(b-a)}{\max(a,b)} \)
where, \( a \) = average intra-cluster distance i.e. the average distance between each point within a cluster. \( b \) = average inter-cluster distance i.e. the average distance between all clusters.

**PCA Visualization**

PCA is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected on the basis of variance that they cause in the output. The feature that causes highest variance is the first principal component. The feature that is responsible for second highest variance is considered the second principal component, and so on. It is important to mention that principal components do not have any correlation with each other [9].
In this section, we discuss the results obtained from approach 1, 2 and 3 which were implemented in the previous section.

6.1 Approach 1: Local Outlier Factor

This algorithm is applied on the structured and template file obtained after parsing the raw data using drain. In Fig 6.2 and 6.3, we can see the plot between the Line Id and the Template Id. As discussed earlier, Line Id indicates the row number and the Template Id indicates the event occurred in that particular row.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>LOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>82.3%</td>
</tr>
<tr>
<td>Recall</td>
<td>0.81</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.77</td>
</tr>
<tr>
<td>Precision</td>
<td>0.75</td>
</tr>
<tr>
<td>Run time</td>
<td>263ms</td>
</tr>
</tbody>
</table>

Figure 6.1: LOF performance

The metrics in Figure 6.1 measure the performance of the model. We can see that the model exhibits an accuracy of 82.3% with a precision of 0.75 that indicates the model have less number of false positives. This means that this is a good choice of algorithm. It also shows that the model has a recall of 0.81 and f1 score of 0.77 which means the model predicted most of the true positives which are considered as outliers.

LOF is used only for outlier detection, it has no predict, Decision and Sample score functions[17]. The LOF score is calculated, if the value of a data point is near to 1 then it is an inlier and the data points which are close to -1 are outliers. The points which are not near to any of the clusters in the plots are considered as anomalies.

We have two values of K for which the data was plotted. We got similar graphs for two of the values.
Figure 6.2: Plot of LineId and TemplateId when K=5

Figure 6.3: Plot of LineId and TemplateId when K=30
6.2 Approach 2: Random Forest

Random forest is applied on the parsed log files and Template file is taken as input. Since the data is time series, random forests does not fit very well for increasing or decreasing trends which are usually encountered when dealing with time series analysis [7]. To remedy this we need to flatten the trends so that the data becomes stationary. The sliding window method is used to analyze data over a finite duration. Later, the log file generated after a failed test is compared with the sequence of the passed test from the same test suite. The log files which are parsed are taken in the form of sliding window and given as input to the model.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>91.02%</td>
</tr>
<tr>
<td>Recall</td>
<td>0.65</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.51</td>
</tr>
<tr>
<td>Precision</td>
<td>0.42</td>
</tr>
<tr>
<td>Run time</td>
<td>357ms</td>
</tr>
</tbody>
</table>

Figure 6.4: Performance of Random Forest

The model will learn the sequence of the parsed log files and based on that it will predict the log file which has a missing sequence or an exceptional sequence. The model is fit on the parsed log files and predicted on the anomalous log file. From Figure 6.4 we can see that the model seemed to be predicting very well with an accuracy of 91%. Then the rows for the wrong predictions are found to analyze whether the anomaly is predicted by the model or not. With an accuracy of 91% it is considered a good approach for this problem. But the F1 score is 0.51 which is very low. This means that the model shows high percentage of false positives. So, the model is not predicting the values are anomalies even though they are not.

Figure 6.5: Plot of Actual and Predicted values on the data
6.3 Approach 3: TFIDF+KMeans+PCA

The tfidf values obtained after performing vectorization using CountVectorizer and transforming the data using TfidfTransformer are plotted using K-Means clustering. After we get the tfidf values, we compare the documents and remove the rows where similar events have occurred and consider the events which shows different values. This is to filter out the normal sequences in the logs. This will help to understand what would be the anomalies from the events that are occurring not frequently. The euclidean distance from the center for the data points can be observed in Figure 6.6. It shows that scatter plot was also plotted to get more clarity for setting the threshold values. We can see that the threshold was set at 0.5 for logs 11900-11909 and 0.6 for logs 8592-8598.

![Visualization of data](image)

Figure 6.6: K-Means Plot for the log files

Then elbow chart is plotted to select the number of optimal clusters for K-means. This is done by using sum of squared error. We can see that the optimal number of clusters is 4, where the we can see an elbow shaped curve in Fig 6.7.

The silhouette score is calculated for our data and it shows which means the clusters are clearly distinguished.

Then the principal components are ranked based on their importance through variance as shown in Fig 6.8, and are selected based on the k value.

Heat Map A heat map is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors [8]. Heat map gives comprehensive overview of how similar logs are and how they distinguished from each other. Figure 6.10 shows the tfidf values for the log files. The logs from 8952 to 8959 are similar and the logs 11900 to 11909 are similar. The dark area in the figure represents similarity and the light area indicates less similarity. With this we can know which logs are not similar and it is useful to filter out the
similar logs as they will have correct sequence of events. Much information about anomalies is found in the area with less similarity.

**PCA** After we filter out the similar events the logs are then compared against each other. For example, the log 8952 and log 8953 are compared and a cluster
plot is drawn to see the differences. As they are same type of logs, they won’t have much differences. Similarly rest of the logs are compared with each other to find out rare events or missing sequence of events that occurred in them. This plot gives information that there are some events that are happening very different from the rest. Later these rows are printed to see what is the difference in the logs in these rows. Below are some plots of the logs compared against each other. Figure 6.11, 6.12, 6.13, 6.14, 6.15, 6.16, 6.17, 6.18 shows the plots of log files. Even though most of the plots are similar some of them have data points which lies away from the clusters. These are considered as outliers and the rows are printed to find anomalous events.

From Figure 6.9, we can see that the model has an accuracy of 93.5%, recall 1, precision 0.91 and f1 score 0.95 which indicates the model is exhibiting highest true positives and negligible amount of false positives which is a very good approach for this research.
6.3. Approach 3: TFIDF+KMeans+PCA

<table>
<thead>
<tr>
<th>Metrics</th>
<th>TFIDF+K-Means+PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>93.5%</td>
</tr>
<tr>
<td>Recall</td>
<td>1</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.95</td>
</tr>
<tr>
<td>Precision</td>
<td>0.91</td>
</tr>
<tr>
<td>Run time</td>
<td>678 ms</td>
</tr>
</tbody>
</table>

Figure 6.10: Performance metrics for approach 3

Figure 6.11: PCA plot for logs 8952 and 8953

Figure 6.12: PCA plot for logs 8954 and 8955
Chapter 6. Results and Analysis

Figure 6.13: PCA plot for logs 8956 and 8957

Figure 6.14: PCA plot for logs 8958 and 8959
Figure 6.15: PCA plot for logs 11900 and 11901

Figure 6.16: PCA plot for logs 11902 and 11903
Chapter 6. Results and Analysis

Figure 6.17: PCA plot for logs 11904 and 11905

Figure 6.18: PCA plot for logs 11906 and 11907
Chapter 7

Discussion

In this section, we discuss the results we got in the previous section. We also interpret how this study is different from other studies.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>LOF</th>
<th>Random Forest</th>
<th>TFIDF+K-Means+PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>82.3%</td>
<td>91.02%</td>
<td>93.5%</td>
</tr>
<tr>
<td>Recall</td>
<td>0.81</td>
<td>0.65</td>
<td>1</td>
</tr>
<tr>
<td>F1 Score</td>
<td>0.77</td>
<td>0.51</td>
<td>0.95</td>
</tr>
<tr>
<td>Precision</td>
<td>0.75</td>
<td>0.42</td>
<td>0.91</td>
</tr>
<tr>
<td>Run time</td>
<td>263ms</td>
<td>357ms</td>
<td>678 ms</td>
</tr>
</tbody>
</table>

Figure 7.1: Performance metrics

In approach 1, we showed some results through plots. The outlier points have also been shown in Fig 5.10. The algorithm was good for finding the outliers that was calculated using reachability distance measure which is used as a distance measure for local outlier factor. It shows an accuracy of 82% making it a good choice of algorithm. It also has less run time which is perfect for larger datasets.

In approach 2, the model predicted the missing or rare sequences in the log file. This model shows an accuracy of 91% which is a very good choice of algorithm. Even though it has less run time, it has very low precision which means that the model has very high number of false positives. This means that the model is predicting the values as anomalies which are actually not. This makes it less likely to be the best approach for this research.

In approach 3, we prioritize the events by using term weighting and the events are given tfidf values based on that. The model groups the Template Ids based on their values which shows exceptional sequence that occurred in a particular row. These are the possible anomalies that causes the system interruption or failure of a particular test suite. After the clustering by using K-Means and PCA, the events that occurred very rarely or the once have been different from the rest of the events are shown in Fig 7.2.

From the PCA cluster plots we can see the outlier events that occur in a log file. These events are now observed in the log file and then differentiated based on the priority. This helps us in finding the anomalous events. Some of these events might lead to the interruption in setting up a connection. This also includes missing and rare sequences in the test logs. For example, in Figure 7.2, EventId with value 0
Figure 7.2: Events having different sequence pattern from the rest is considered as an anomaly as it indicates rare sequence which was mentioned in Chapter 5.

This would be helpful for the developers for finding out where the event has been occurring and they need not manually go through the log to find out what is the reason behind system interruption or misconnections. This model works better for this project as it is able to distinguish and clearly mention where’s the problem is occurring without manually looking at the log file. And most of the developers won’t be able to go throughout the log file every time there’s a problem. So, this would actually make their work easier. It also shows very good performance with an accuracy of 93% and F1 score of 0.95 which makes it best to work for this research. Even though it has a bit high run time compared to the other approaches it has less run time in real.

Apart from the common performance metrics we also plotted a heat map and calculated silhouette index to get more insight on the results obtained. From heap map we know the similarity of the test logs that were taken. This helps us simplify the process of finding which logs are not similar. From the silhouette index we know how good the clustering technique that we implemented. We got a silhouette score of 0.92 which indicates that the clustering technique detected to find anomalies is very good. Based on the above, after experimenting which model gives better results for this research TFIDF with K-Means and PCA clustering seems to work well with this problem, which gives an answer to RQ2.

In the paper [36] they proposed a framework for online log anomaly detection and diagnosis using a deep neural network based approach. encodes entire log message including timestamp, log key, and parameter values. This was done only using LSTM. Even though LSTM works better the approach, it typically takes more than 30 minutes to run for this type of data. Whereas, my study used various approaches and compared the results to find the optimal solution among them. Only the log event is considered in my approach to find anomalies which makes it much easier to implement the models that had less run time.

In [44], the problem of rather high amounts of false positives that all anomaly de-
tection techniques suffer from remains unsolved. The approach 3 that I mentioned in this study would contribute to this as the model has a good percentage of true positives which reduces the problem of having high number of false positives.
Chapter 8

Limitations and Challenges

The challenges faced during this project and the limitations of the project are discussed in this section.

- Data analysis, which means understanding the data and finding the right method to do all the pre-processing and is a bit tough. As the data is in its raw form analyzing the data was harder. There were several unwanted text, unwanted symbols and null entries and sections of the dataset that were not fit to be used for the intended purpose of this project.

- As the data is unlabeled and it’s in raw form, finding the right method for parsing was difficult. This needed a lot of research and understanding of the existing parsing methods.

- Since data can behave in an unpredictable manner, using the right method to predict was challenging. In approach 1, it calculates the TemplateId distance rather than the outlier distance, and in approach 2 the model does not predict the anomaly that was added. Choosing the next method to perform on this data was tough.

- High dimensional data is very difficult to visualize. Selecting the right dimensionality reduction technique is still an unsolved problem in Machine Learning. This was a challenge in this project as well.

8.1 Validity Threats

8.1.1 Internal Validity

- Data analysis and parsing of the data have been made to work on the raw and sparse data. While parsing, we might sometimes miss the important information that would help us in this project.
  We managed to overcome this threat by carefully parsing the raw data under the guidance of supervisor at Ericsson using Drain parser which gives two files that contain all the information from the logs except the data that was removed using regex code. This helped to get information that is more useful for applying the machine learning algorithms.

- The algorithms used in this study have different metrics for evaluation. They do not have metrics in common to compare. This was a major threat in this
Even though the algorithms have evaluation metrics in common, they were evaluated using individual metrics and the algorithm which is able to detect the anomalies from the log files is stated as best performing algorithm. The rest two algorithms were not accurately able to detect the anomalies.

8.1.2 External Validity

Since the study is done based upon the interests of the people at Ericsson, this would actually help them in overcoming some real problems. So, external validity won’t be applicable in this situation as the project is done based on real world data.
Chapter 9

Conclusions and Future Work

This study helps in finding an optimal solution for the anomaly detection problem in various and large log files that has been a problem. Though there have been many researches and studies performed based on this, this project covered studied most of the possible ways that could address this problem. As, RQ1 was answered by doing a literature review, RQ2 was done by conducting an experiment on the various machine learning models that could fit the best and give a better solution to this problem. TFIDF with K-Means and PCA clustering seems to be an optimal solution for this project. Even though this gave us good results, deploying this method for real usage was a bit challenging. This is because while doing the project there was a lot of manual work that was done.

First, applying the suitable parsing method so that there won’t be any loss of useful information from the log files. Second, the test logs are studied to take only the part of the log that is actually useful for this, as the log data contains each and every information sometimes which might not be useful or does not have any information regarding the anomalies. Taking this information gives only the false predictions of the normal data which is even more disastrous. But, suitable parsing technique and the machine learning models were chosen after a careful literature review and under the guidance of supervisors at Ericsson.

As future work, developing a method or tool that might actually take the useful part of the log files and then prioritize based on whether the log file which is generated after successful execution of hardware or software tests. Providing a solution on how to avoid the anomalies occurred after finding where they have caused would be a better improvement for this work.
Bibliography

[1] “6.2. feature extraction — scikit-learn 0.23.2 documentation.”
[6] “Fig. 10.1 cartoon representation of a random forest classifier, researchgate. https://www.researchgate.net/figure/cartoon-representation-of-a-random-forest-classifier-fig137361248(accessed sep. 05, 2020).”
[17] “Outlier detection with local outlier factor (lof) — scikit-learn 0.23.2 documentation.”


