CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Noise Handling for improving anomaly detection in application logs

Master's thesis in Computer science and engineering

ZHIYAN ZHANG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

NOISE HANDLING FOR IMPROVING ANOMALY DETECTION IN APPLICATION LOGS

ZHIYAN ZHANG

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Noise Handling for improving anomaly detection in application logs

ZHIYAN ZHANG

Supervisor: Regina Hebig, Department of Computer Science and Engineering
Advisor: Juliana Furuzato, Volvo Group IT
Examiner: Lucas Gren, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Noise Handling for improving anomaly detection in application logs

ZHIYAN ZHANG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Application logs can often play a vital role in maintaining information systems. However, many of the applications in production environments are generating more and more logs. This can make it particularly difficult for users to locate the needed information in a short time. As anomalies can appear in the application logs and the symptoms of the problems may lead to some serious consequences such as huge economic loss, it is crucial to detect them effectively. The cases that successfully apply machine learning to solve problems in the software engineering area are gradually increasing, for example, the end-to-end pipelines became more automated after applying some artificial intelligence algorithms. Machine learning can also be advantageous in detecting anomalies in application logs. In this thesis study, we aim to investigate whether attribute noise removal has an impact on improving the learning performance of anomaly detection. We achieve our goal by applying an existing machine learning algorithm, log clustering, to identify anomalies. Also, an existing attribute noise removal method, PANDA, is used to study the effect of attribute noise removal on the learner performance of anomaly detection. Then we evaluate whether attribute noise removal can help improve the anomaly detection process by comparing the training results of different experimental groups. Overall, as the percentage of attribute noise removal increases, the values of precision, F1-Score, and MCC increase slightly. We conclude that attribute noise removal has the potential to be beneficial to the anomaly detection process, such as increasing the number of detected false cases.

Keywords: Software engineering, Anomaly detection, Attribute noise, Log Clustering, PANDA.

Acknowledgements

First of all, I would like to express my heartfelt thanks to all the people who have ever supported me in this thesis work.

I would like to express my gratitude to my supervisor, Regina Hebig at Chalmers University of Technology for giving me suggestions and encouragement all through my writing. It has been greatly joyful to study under her guidance and supervision.

I am also extremely grateful to my advisor Juliana Furuzato for giving me the opportunity to work on this thesis, helping me understand the logs and always getting back to me quickly.

My examiner Lucas Gren gave me valuable advice regarding this thesis paper, so I would also like to thank him.

In addition, my warm gratitude also goes to the interviewees who helped me understand deeply the expectations from the company.

Finally, many thanks to my family and friends for whose support benefited me during the thesis study.

Zhiyan Zhang, Gothenburg, January 2023

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Problem Description . . . 1
1.1.1 Research questions . . . 2

2 Background **3**
2.1 Anomaly . . . 3
2.1.1 Anomaly detection algorithms . . . 3
2.1.1.1 Log Clustering . . . 4
2.2 Log Parsing . . . 5
2.2.1 Drain . . . 5
2.3 Feature Extraction . . . 6
2.3.1 Session Windows . . . 6
2.3.2 TF-IDF . . . 7
2.4 Noise . . . 8
2.4.1 Attribute Noise . . . 8
2.4.2 Noise handling approaches . . . 8
2.4.2.1 PANDA . . . 9
2.4.3 Noise handling for anomaly detection . . . 11
2.5 Evaluation . . . 11
2.5.1 F1-Score . . . 11
2.5.2 MCC . . . 12

3 Method **13**
3.1 The data sets . . . 13
3.1.1 The Volvo log data set . . . 13
3.1.2 Public log data set . . . 14
3.2 Experimental set up . . . 14
3.3 Setting Up the Tool Chain . . . 16
3.3.1 Log collection . . . 16
3.3.1.1 Collecting Volvo log data . . . 16
3.3.1.2 Collecting public log data . . . 17
3.3.2 Log Parsing . . . 18
3.3.2.1 Parsing the Volvo log data . . . 18

3.3.2.2 Parsing public log data 19
3.3.3 Feature Extraction 20
3.3.3.1 TF-IDF 21
3.4 Noise Handling 22
3.5 Anomaly Detection 24
3.6 Evaluation 25
3.7 Interviews 26

4 Results and Discussions **29**
4.1 Anomaly Detection 29
4.1.1 Baseline measures for anomaly detection: good and bad configurations 29
4.1.2 Group One: Anomaly Detection without Handling Attribute Noise 31
4.1.3 Group Two: Anomaly Detection with Handling Attribute Noise 32
4.2 Interviews 33
4.2.1 Discussion of Results in Context of Interviews 34
4.3 Limitations and delimitations 35

5 Conclusion and future work **37**

List of Figures

2.1 Session Windows Grouped by Timeout Size and Max Duration Size . 7
2.2 Session Windows Grouped by Partition Key 7
2.3 The main procedure of PANDA [1] 10

3.1 The Overview of Methodology 13
3.2 The Company's Logs Example 14
3.3 HDFS Logs Example 14
3.4 HDFS Data Labels Example 14
3.5 Anomaly Detection Process in Group 1 15
3.6 Anomaly Detection Process in Group 2 15
3.7 Process of Log Collection 16
3.8 Log Parsing Results 19
3.9 Event Templates 20
3.10 Structured Data 20
3.11 Session Windows Example 21
3.12 TF-IDF Calculation Example 22
3.13 Noise Handling Example 23
3.14 Processes of Log Clustering 24

4.1 Evaluation Scores of Max Distance 30
4.2 Evaluation Scores of Distance Threshold 31
4.3 Evaluation Scores of Init Samples 31

List of Tables

3.1 Regular expressions and Functions 19
3.2 Background of Interviewees in this Study 27

4.1 Values of Parameters in Group G and Group B 31
4.2 Results in Group One 32
4.3 Results in Group Two 32

1
Introduction

Machine learning algorithms have been successfully used to deal with big data. With the help of these algorithms, large amounts of data can be processed efficiently. The results automatically analyzed by these algorithms can be helpful in many ways, such as decoding patterns and predicting trends[2].

One typical example of applying machine learning to big data is using machine learning algorithms to conduct log analysis. In the application logs, a large amount of data is generated every minute. Sometimes, the log data will be checked only when there is a problem with the application and the checking method is mostly manual. Some unnoticeable anomalies will not only affect the system operation, and in severe cases, the application may crash because of not noticing them in time, but also cost developers a lot of time to troubleshoot errors by manually checking application logs. What's more, the data in application logs varies from application to application because of the different formats, development styles, and so on [3]. Therefore, it is difficult to build a static analysis tool for logs. Some forms of machine learning are required to better analyze log data.

Sometimes, however, not all errors in the server are worth the developer's time to focus on. Some errors may simply be reported when the server was temporarily disconnected due to the network restart. Other errors may simply be the result of developers trying to write their own programs to test the code. These are errors that have little to do with the server itself and are called "ignorable errors". Therefore, it would make a lot of sense to be able to automatically determine which errors in the log are the ones that are truly noteworthy, namely anomalies, before they cause huge damage and impact. To solve this problem, some techniques for anomaly detection can be adopted.

1.1 Problem Description

In today's era of big data, more and more data sources appear, one of which is enterprise application logs. Application logs play a key role in troubleshooting errors. To ensure that the system is running properly, the developers involved are responsible for periodically checking the system application logs. Anomalies in application log data are considered to be patterns or characteristics which do not follow the average or normal behavior during perfect operations [4] [5]. In the context of this

study, anomalies refer to the errors that really need to be paid attention to. Traditionally, they have had to check for log anomalies manually, using keyword searches and so on. However, with the advancement of technology, those industrial systems are increasing in size and complexity, making it difficult to manually check for log anomalies, which is therefore very time-consuming and ineffective [6]. In this case, a method to automatically detect anomalies is needed.

Noise in the data refers to irrelevant or meaningless information when analyzing data [7]. Noise may affect the quality of anomaly detection negatively. However, little is known about the extent of this effect and the improvement potential that noise handling can have on anomaly detection. Typically, two types of noise are distinguished: class noise and attribute noise [8] [9]. The former can come from contradictory class labels or misclassifications in the data [10]. Attribute noise is the errors that are introduced in the attribute values of the instances [10]. But the noise of any kind can have a significantly bad effect on any kind of data analysis [7]. It could lead to decreased accuracy, increased training time, and increased model complexity. This study will look into using noise handling to improve anomaly detection.

1.1.1 Research questions

The primary goal of this thesis is to improve anomaly detection in application logs by handling noise. Also, in the data set that we are going to use, the number of anomalies is small. Therefore, we will focus on unsupervised anomaly detection algorithms. Due to that, we will also focus on attribute noise only. Thus, the research question is:

RQ: How can we improve the predictive performance of a learner for anomaly detection in application logs by handling attribute noise?

2
Background

In this section, relevant theoretical knowledge and related work are presented to readers, in order to help readers get a better understanding of the whole process. We first introduce the definition of anomaly. After that, the relevant theory of technique used in each step during the anomaly detection process is presented. They are techniques for log parsing, feature extraction, noise handling, and evaluation.

2.1 Anomaly

Anomaly in data analysis refers to rare items, events, or observations that raise suspicions by differing significantly from the majority of the data [11]. The anomaly can cause an error but is not equal to an error. Anomaly detection methods based on logs has become an important research topic in academic and industry area [6]. Typically, techniques used for anomaly detection include distance-based, clustering-based, density-based and classification-based algorithms [12]. Distance-based algorithms identify anomalies by computing distances between the queries; [13], one common algorithm is *KNN*. Clustering-based algorithms work by grouping data points. The data points in the same group are similar in properties. One example algorithm is *K-Means Clustering* [14]. Density-based algorithms, for example, *DBSCAN*, computes clusters based on a threshold for the number of neighbors, *minPts*, within the radius ε (with an arbitrary distance measure) [15]. Classification-based algorithms such as *Random Forest* [16] classify the data into many categories, such as normal and abnormal.

2.1.1 Anomaly detection algorithms

Anomaly detection is more and more widely used in the area of software engineering. Hangal and Lam [17] introduce a practical tool DIDUCE that aids programmers in finding complex Java program errors and identifying root causes by automatic anomaly detection. The tool works with Java byte codes and extracts invariants dynamically from program executions by using the concept of dynamic invariants and checking. Their experimentation with four complex Java projects shows that DIDUCE is effective.

Baah et al. [18] present a machine learning technique that performs anomaly detection as software is executing in the field. They use a fully observable Markov model, in which parameters are estimated using the Baum-Welch algorithm [19].

This technique allows faults with anomalies to be found and fixed before they wreak havoc on the system. They applied their novel technique in two case studies on a mid-sized Java program. The results show their approach is promising.

Two typical machine learning types of anomaly detection algorithms are supervised and unsupervised. Supervised learning refers to the process of adjusting the parameters of a classifier to achieve the desired performance using a set of samples of known classes. One supervised algorithm is *Random Forest* [16] which classifies data by using decision trees.

Unsupervised learning techniques solve problems in pattern recognition based on unlabeled training samples. One example technique is *Density Based Anomaly Detection* which utilizes local outlier factors to investigate the density [3]. An object is identified as an outlier in the case that its density is lower than its neighbors'.

Grover [4] elaborate on three reasons why anomaly detection for application log data is challenging, whether automated or done manually. These reasons are unstructured plain text, redundant run-time information, and extensive unbalanced data. They conducted four experiments using these unsupervised learning techniques: *K-Means Clustering*, *K-NN Global Density based*, *LSTM Neural Network* and *LSTM Auto Encoder(LSTM-AE)*. They found *LSTM-AE* scored the highest accuracy.

2.1.1.1 Log Clustering

LogCluster [20] is a clustering-based method that can help resolve some log-based problems. It has already been successfully used in different production environments in Microsoft for many years. One of the success stories is that LogCluster was used by a Microsoft service team to work with Active Monitoring. The two tools together can help detect more problems and recover the service in a shorter time. In addition to applying in Microsoft, LogCluster has a high value in research. Many of the anomaly detection methods developed later were compared to this tool. PLELog [21], Probability Label Estimation Log, is an example. This semi-supervised log-based anomaly detection method was developed recently and was evaluated by comparing it with other valuable approaches including LogCluster.

LogCluster will first weight those log events by utilizing the combination of two methods: IDF-based Event Weighting and Contrast-based Event Weighting. In IDF-based Event Weighting, the more frequently the events occur, the lower weights they will have. In Contrast-based Event Weighting, the events that occur only in the production environment are weighted higher than those occurring in both the lab and production environment. The log sequences that contain multiple unique events are then vectorized into event count vectors. Second, in the clustering step, the Agglomerative Hierarchical clustering technique [22] will be performed after computing the similarity (2.1) [20] of any two log sequences. In the Equation (2.1), S_i and S_j are two N-dimensional vectors and S_iE_k is the K^{th} event in the j^{th} vector [20].

$$\begin{aligned}\text{Similarity}\left(S_i, S_j\right) &= \frac{S_i \cdot S_j}{\|S_i\| \, \|S_j\|} \\ &= \frac{\sum_{k=1}^{n} S_i E_k \times S_j E_k}{\sqrt{\sum_{k=1}^{n} \left(S_i E_k\right)^2} \times \sqrt{\sum_{k=1}^{n} \left(S_j E_k\right)^2}}\end{aligned} \tag{2.1}$$

The log sequences will be clustered together based on the distance metric between the clusters defined by users. Another parameter, distance threshold θ, decides when to stop clustering. The distance metric is actually affected by the maximum distance between the log sequences in the clusters. When the maximum distance between two clusters exceeds the distance threshold, the clustering process between the two clusters will stop. Lastly, LogCluster selects a representative log sequence in each cluster by computing the centroid of the cluster. The computation of each log sequence's score is shown in Equation (2.2) [20]. It is on the basis of the sequence's average distance to other log sequences in the same cluster. In the equation, i is a log sequence. n is the number of log sequences in the cluster. The log sequence scores minimum will be selected as the representative [20].

$$\text{Score}(i) = \frac{1}{n-1} \sum_{j=1}^{n} \left(1 - \text{Similarity}\left(S_i, S_j\right)\right) \tag{2.2}$$

2.2 Log Parsing

In general, application logs are unstructured by default. It is hard and time-consuming to find detailed information the developer needs within unstructured logs[23]. Thus, it is imperative to structure the format of the logs.

Log parsing is basically the process of splitting log data into small chunks of information that are easier to manipulate and store. Then the small chunks are placed into specifically named fields by following a set of rules [24]. There are diverse application scenarios of log parsing, such as in the area of performance modeling [25]. In a use case that Facebook reported several years ago, all log event templates are extracted from the log data to be input to validate potential performance modeling improvements [26]. Another example where log parsing is widely used is anomaly detection in logs. The first step in anomaly detection is usually log parsing. The parsed log serves as an essential data source for detecting anomalies. Some methods of log parsing that have been studied include deep learning [27], invariant mining [28], and so on.

2.2.1 Drain

Drain is an efficient online parsing method developed in recent years and has been widely applied. For example, Aussel et al. [29] put drain into use to improve performances of log mining for anomaly prediction. Meng et al. [30] developed an

adaptive log parsing method based on drain and other log parsing techniques.

Compared to offline log parsers, one advantage of drain is that drain can parse logs in a streaming manner. Drain is fixed depth tree based. With fixed depth, it could group the logs more effectively since it limits the number of log groups against which the raw log message needs to be compared [31]. Also, drain utilizes regular expressions to simplify the log data. An example to explain how it simplifies logging data: there are two log messages, one message includes "connect to server id: 1234" and another includes "connect to server id: 5678". Drain would parse these two messages into one template which is "connect to server id: <*>".

2.3 Feature Extraction

It is common for large data sets to have hundreds or thousands of features in the data. Whereas not all the features are of great importance for training the model. Thus, reducing the number of features and converting the valuable features into a format that can be used for modeling can be very helpful for applying machine learning models. In the context of the data format being words, feature extraction helps convert words into numeric vectors. In this way, the data could be better understood while modeling.

2.3.1 Session Windows

For streaming and unbounded data, windowing is a substantial technique to deal with such data [32, 33]. Session windows is one of the major types of windowing and a great technique for feature extraction.

It mainly has three parameters: *timeout size*, *max duration size* and *partition key* [34]. *Timeout size* is the gap size of the windows. Events occurring within the timeout size will be grouped together. If no events occur within the timeout size, the session window will be closed at the timeout. If events continue occurring within the timeout size, the window will be continuously extended until reaching the *max duration size*. Figure 2.1 shows an example. Assume that *timeout size* is 5 and *max duration size* is 8. A spate of events $E1$, $E2$ and $E3$ occur. The window starts from the first event $E1$. When $E2$ occurs, it is still within the *timeout size* since $E1$ occurs. So $E1$ and $E2$ are in the same window. But $E3$ is not in the same window as $E1$ and $E2$ due to the *max duration size*. Although $E3$ also occurs within five minutes from $E2$, there is a checkpoint at the 8-minute mark. Window 1 will be closed at the checkpoint and a new window will start from $E3$.

Another way to group session windows is based on the *partition key*. Events with the same key will be grouped together. It is not time-based. For instance, as shown in Figure 2.2, events $E1$ and $E3$ have the same $key1$ and will be grouped in a session window. $E2$ will be in another session window.

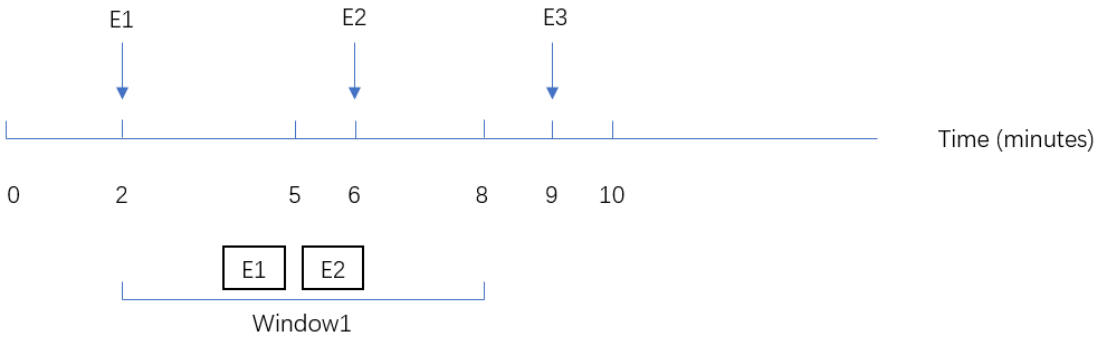

Figure 2.1: Session Windows Grouped by Timeout Size and Max Duration Size

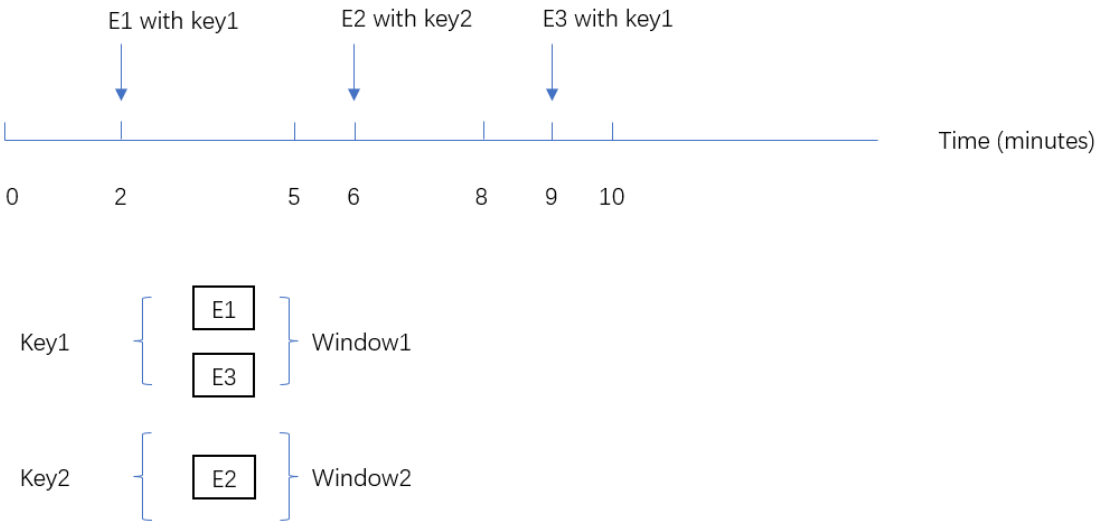

Figure 2.2: Session Windows Grouped by Partition Key

2.3.2 TF-IDF

Term weighting is a process carried out during text indexing to assess the importance of each term [35]. TF-IDF is one of the most popular term weighting schemes[36]. It stands for Term Frequency–Inverse Document Frequency. TF, Term Frequency, is the standard weighting and it denotes the frequency a term appears in a document. The importance of a term increases proportionally with the number of times it appears in the document, but decreases inversely with the frequency of its appearance in the corpus. IDF, Inverse Document Frequency, refers to global weighting. The IDF of a particular term can be obtained by dividing the total number of documents by the number of documents containing the word and taking the logarithm of the quotient obtained. Following are the equations of calculating TF, IDF, and TF-IDF. In Equation (3.1), the denominator is increased by 1 to avoid a zero denominator. This means none of the documents contain that term in this case.

$$TF = \frac{The\ number\ of\ times\ a\ term\ appears\ in\ the\ document}{The\ total\ number\ of\ terms\ in\ the\ document} \tag{2.3}$$

$$IDF = \log(\frac{The\ total\ number\ of\ documents\ in\ the\ corpus}{(The\ number\ of\ documents\ containing\ the\ term) + 1}) \tag{2.4}$$

$$TF - IDF = TF \times IDF \tag{2.5}$$

2.4 Noise

The quality of a data set can usually be described by two sources of information: (1) attributes and (2) class labels [8]. However, data in the real world is usually not so perfect. Data sets often contain data with meaningless information. This kind of data is called noisy data. The noise in the data is differentiated into two categories: class noise and attribute noise. Class noise can occur when training the data set in supervised learning, since the training data has both features and labels. While attribute noise is potentially relevant for all forms of anomaly detection learning methods.

2.4.1 Attribute Noise

Attribute noise means corruption of the value of the attributes in the data. It may occur when selecting irrelevant, redundant, or empty attribute values for training [8, 37]. One approach to handle this is using a dual-voting-based learning paradigm [38]. In the paradigm, a classification and regression tree (CART) model is adopted as the generic classifier to evaluate the performance of training data sets. Another method to solve the problem of attribute noise is using the pairwise attribute noise detection algorithm (PANDA), which identifies training instances with a significant deviation from normal given the values of a pair of attributes [1].

2.4.2 Noise handling approaches

In the software industry, the most recent trend is the integration of artificial intelligence capabilities based on advances in machine learning [39]. One of the domains, noise handling, has been studied for diverse learners and tasks in software engineering.

To handle class noise, some proposed solutions such as *polishing*, *filtering* and *robust* could be helpful. *Polishing* exploits the interdependence between the components in the data set to identify noise elements and suggest appropriate replacements. *Filtering* removes misclassified instances. *Robust* means that a certain amount of noise can be toleranced. For the problem of attribute noise, using PANDA to identify instances that deviate largely from the normal ones can help solve it [1].

Zhu et al. [40] applied a class noise handling technique to help improve the process of making informed logging decisions. The technique uses *KNN* algorithm to identify the noises and SMOTE [41] to help eliminate the bias. SMOTE is an over-sampling technique that can balance the data by creating synthetic logged instances. Results show that removing noise data can help make the process of learning the common logging knowledge more effective.

Teng used a decision tree learning algorithm and evaluated the performance of three noise handling methods separately [42]. The data comes from the UCI Repository [43] of machine learning databases. He collected twelve data sets from all walks of life. They are about cars, lenses, lung cancer, and so on. He investigated the following methods for handling both class and attribute noise: *robust*, *filtering*, and *polishing*. He concluded that both *filtering* and *polishing* could be helpful for solving imperfections in the data although *polishing* scored higher accuracy than *filtering*.

Al-Sabbagh et al. [16] did research on improving test case prediction by handling class and attribute noise. They coped with class noise by relabeling repeated code lines that came with different class values and used an existing elimination-based approach called PANDA[1] to address the problem of attribute noise. For evaluation, they selected *the Random Forest* model and compared the result by examining the precision, recall, and f-score. At last, they found handling class noise could improve the performance of the learner, while eliminating attribute noise almost had almost no improvement on the performance. This study can not show that addressing attribute noise is useless in every domain. Thus, the question needs to be answered whether it can be beneficial for anomaly detection in application logs.

From these studies, it can be seen that handling noise problems has varying degrees of benefit in different domains.

2.4.2.1 PANDA

So far, very little attention has been paid to attribute noise compared to class noise. And few methods can be used to detect instances with attribute noise, mainly due to the high complexity of the problem [1].

PANDA is one of the novel attribute noise handling methods. Its full name is *Pairwise Attribute Noise Detection Algorithm*. In the decade since it was developed, it has only been used in several research fields. Phillips [44] modified PANDA to detect contamination in surface electromyography (SEMG) signals. SEMG has been widely used in different areas such as athletic training and diagnoses of neuromuscular diseases [45, 46]. In Phillips's work, PANDA is modified and then used as a classifier to classify whether the data is clean or contaminated. The result of classifying by using PANDA is compared with the one that uses SVM. PANDA outperforms SVM because PANDA can detect the noise, which levels are too small that SVM can not detect.

Procedure: Calculate Noise Factor(string FUNC, dataset X)
input: Dataset $X = [x_{ij}]_{n \times m}$ with n observations and m attributes, where x_{ij} is the value of the j^{th} attribute for the i^{th} observation. x_{*j} denotes the j^{th} attribute.
output: Noise Factor S_i for instance i, $1 \leq i \leq n$.

1. $s_{ikj} = 0 \ \ \forall\, i = 1, \ldots, n$ and $j, k = 1, \ldots, m$ //initialize values
2. for $j = 1$ to m
3. Transform attribute x_{*j} into the partitioned attribute $\hat{x}_{*j} \in \{0, \ldots, L-1\}$ where $L = \#$ of bins
4. for $k = 1$ to m
5. if $k \neq j$
6. calculate $Mean(x_{*k} \mid \hat{x}_{*j} = l)$ and $Std(x_{*k} \mid \hat{x}_{*j} = l)$ for $l = 0, \ldots, L-1$
7. for $i = 1$ to n
8. $s_{ikj} = |x_{ik} - Mean(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij})| \ / \ Std(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij})$
9. //s_{ikj} is the standardized value of the attribute value x_{ik} for the i^{th} observation given the partitioned attribute value $\hat{x}_{ij}$
10. endfor
11. endif
12. endfor
13. endfor
14. if FUNC='SUM'
15. $S_i = \sum_{k=1}^{m} \sum_{j=1}^{m} s_{ikj}$
16. endif
17. else if FUNC ='MAX'
18. $S_i = \max_{\{j=1,\ldots,m\}\{k=1,\ldots,m\}} \left\{ s_{ikj} \right\}$
19. endif
20. return S_i

Figure 2.3: The main procedure of PANDA [1]

Given a pair of attribute values, PANDA detects noise by identifying instances with significant deviations from normal. For example, if one of the attributes is close to the normal value while the other is far away from normal. The attribute which value deviates largely from the normal value is much more likely to be considered noise. In the PANDA algorithm, each instance is iterated over and divided into a number of bins according to the set bin size. If the property has no binding value, each bin will have the same number of instances. Then those with the same attribute value will be divided into the same partition. After the partitioning process is complete, each bin's mean and standard deviation are calculated. Each instance of the attribute x_k generates a standardized value [16]. The normalized value is then calculated by subtracting the ratio of bin's mean to standard deviation relative to x_j from each instance value in x_k [16]. All attribute pairs will go through these steps. Then, SUM or MAX is calculated for each observation. Instances with large SUM or MAX values are considered noise. The algorithm eventually outputs a list of instances sorted by noise likelihood from highest to lowest. The pseudo-code of the main procedure of PANDA is shown in Figure 2.3. Suppose m is the number of attributes in the data set and n is the number of instances. The time complexity of this algorithm is $\mathcal{O}(m^2 n)$.

2.4.3 Noise handling for anomaly detection

Many studies are showing that solving noise problems can help improve anomaly detection to some extent.

Zhong et al. [47] devised a graph convolutional label noise cleaner to handle the class noise in order to train a better classifier for anomaly detection. The method they adopted to devise the label noise cleaner is Electromagnetism-like(EM) optimization mechanism. The EM algorithm is a population-based stochastic global optimization algorithm [48]. The experiments they conducted proved its effectiveness and versatility.

Kroll et al. [49] found that noise in the data could cause their anomaly detection algorithms to fail to identify abnormal behavior for this specific state. They tackled this problem by using dynamic thresholds.

Kini and Madakyaru [50] performed multi-scale decomposition of the data using wavelet functions and data reconstruction techniques to handle the noise. Their purpose in dealing with noise is to improve anomaly detection that captures the departure event of a variable from the usual behavior with the use of fault indicators.

This study will focus only on attribute noise handling for anomaly detection, since most anomaly detection algorithms are unsupervised and class noise is irrelevant to them.

2.5 Evaluation

Evaluation techniques are vital, since they help us to learn about the quality of the anomaly detection model that is trained with and without noise handling.

2.5.1 F1-Score

F1-Score is one of the most important evaluation metrics in machine learning. Risch and Krestel [51] utilized F1-Score to evaluate the model for robust aggression identification. Lee and Shin [52] use F1-Score to assess the learning performance of the model that was trained on clinical benchmark data.

F1-Score is defined as the harmonic mean of precision and recall. It ranges from 0 to 1. The higher the score, the better the model is. F1-Score takes both precision and recall into account. Precision refers to how many of the detected anomalies are true anomalies. Recall means how many anomalies are detected as true anomalies. The values of precision and recall are calculated by true positive(TP), true negative(TN), false positive(FP), and false negative(FN) as shown in Equation (2.6) and (2.7). The calculation of F1-Score can be seen as Equation (2.8).

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.8}$$

2.5.2 MCC

MCC is a special case of the phi coefficient ϕ [53]. The full name of MCC is Matthews Correlation Coefficient. Saqlain et al. [54] apply MCC as a feature subset selection metric to the study of diagnosing heart disease. Jones and Ward [55] use MCC to predict disordered regions in proteins.

MCC considers true class and predicted class as two binary variables and calculates their correlation coefficients. The higher the correlation between the true value and the predicted value, the better the prediction. The range of MCC score is from -1 to 1. Score 1 is the best value which means the predicted values are exactly the same as the actual values. Score 0 means the prediction is no better than a random prediction. Contrary to 1, -1 stands for the worst value which means the predicted values are perfectly wrong. Similar to precision and recall, the value of MCC is calculated based on true positive(TP), true negative(TN), false positive(FP), and false negative(FN). The formula is presented as Equation (2.9).

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.9}$$

3
Method

The goal of this study is to examine the effect of handling attribute noise in application log data for improving anomaly detection. This section describes how the anomaly detection process would be conducted and the noise handling method would be applied. The overview of the methodology is shown in Figure 3.1.

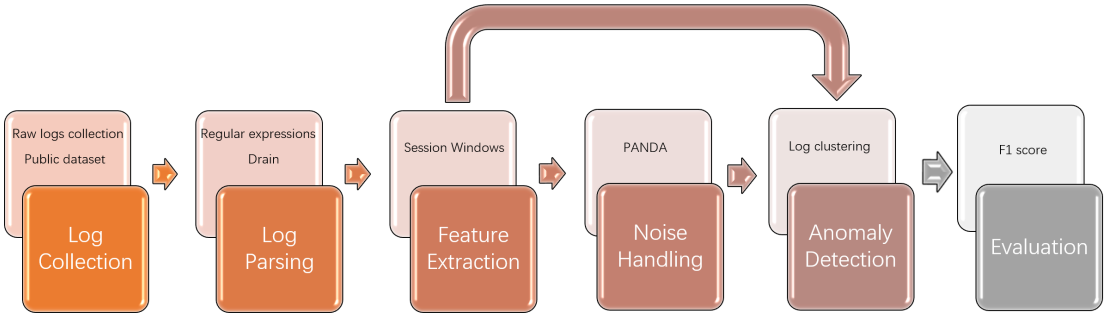

Figure 3.1: The Overview of Methodology

3.1 The data sets

Two datasets used in this study are described here. One is from Volvo. The real Volvo data can not be shown in this paper due to confidential issues. The other is an open-source data set. The examples in this paper about the data from this public data set is real.

3.1.1 The Volvo log data set

The original data is collected from a large Swedish industry, Volvo, that has a large number of application logs. The data set consists of logs generated by four servers over a period of six months. The size of the whole data set is approximately 100 GB. The format of the data set is *xml*. The attributes that the logs have in those four servers are:

$[Timestamp]\ [Category][AppDomain][Severity][Message][Properties]$

The logs are chain event logs. This means there are many logs that are correlated together. What ties these correlated logs together is their correlation id. Logs belonging to a chain have the same correlation id. Figure 3.2 shows a sample of the company's data. The data in the sample is not real data. In this sample, the two log events belong to one set of chain event logs.

```
<Log>
<Event Timestamp="2021-02-23T15:01:18+07:00" Category="System" AppDomain="Product" Severity="Error"><Message><![CDATA[Message Message2]]></Message>
                <Properties><P N="CorrelationId"><V><![CDATA[2wwwwwww-a333-4000-8afa-999999e299f5]]></V></P><P
N="RequestingNodeId"><V><![CDATA[company.server@ASDFGHKKL000200]]></V></P></Properties></Event>
<Event Timestamp="2021-02-23T15:01:23+07:00" Category="System" AppDomain="Product" Severity="Verbose"><Message><![CDATA[Message Message4]]></Message>
                <Properties><P N="CorrelationId"><V><![CDATA[2wwwwwww-a333-4000-8afa-999999e299f5]]></V></P><P
N="RequestingNodeId"><V><![CDATA[company.server@QWERTYUIO000199]]></V></P></Properties></Event>
</Log>
```

Figure 3.2: The Company's Logs Example

3.1.2 Public log data set

The public data set is generated from the Hadoop Distributed File System [56] (HDFS). This file system provides high-throughput access to application data. The logs are generated in a private cloud environment and are written directly into local disks. The size of the data set is around 1.5 GB. In this system, when a block is allocated, modified, or deleted, there would be various log messages generated about that block. Similar to the Volvo data, the logs related to a block have the same block id(BID). As shown in Figure 3.3 and Figure 3.4, each log has a block id. Each block id is manually assigned a label by the original authors in a separate file. The types of labels are *Anomaly* and *Normal*.

```
081109 203518 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.jar. blk_-1608999687919862906
081109 203519 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010
081109 203519 145 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50010
081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-1608999687919862906 terminating
081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-1608999687919862906 terminating
081109 203519 145 INFO dfs.DataNode$PacketResponder: Received block blk_-1608999687919862906 of size 91178 from /10.250.10.6
```

Figure 3.3: HDFS Logs Example

BlockId	Label
blk_-1608999687919862906	Normal
blk_7503483334202473044	Normal
blk_-3544583377289625738	Anomaly
blk_-9073992586687739851	Normal
blk_7854771516489510256	Normal
blk_1717858812220360316	Normal
blk_-2519617320378473615	Normal

Figure 3.4: HDFS Data Labels Example

3.2 Experimental set up

To evaluate the improvement of performance of anomaly detection with noise handling, an experiment was designed. Two experimental groups were set up. The

common process in these two groups is *Setting Up the Tool Chain*. The steps in this common process are *Log collection*, *Log Parsing* and *Feature Extraction* as shown in Figure 3.5 and Figure 3.6. All the settings of this process in the two groups are same. First, the Volvo data set and the HDFS data set are collected. Second, the log data in these two data sets is parsed. Then the parsed data from two data sets is converted to numerical vectors.

The steps after *Feature Extraction* are a little different in these two groups. In group one, the anomaly detection process was performed without handling noise. After *Feature Extraction* step, the anomaly detection algorithm was applied directly to the preprocessed data.

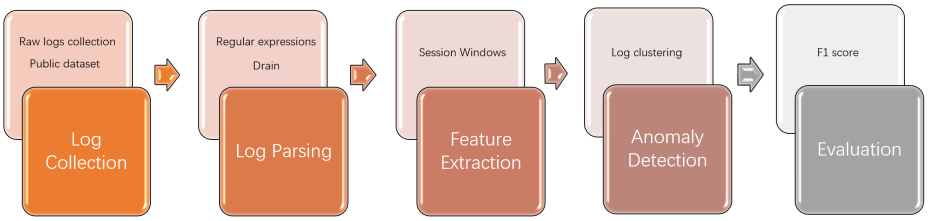

Figure 3.5: Anomaly Detection Process in Group 1

In group two, the anomaly detection process was performed with handling noise. The noise handling approach was run after *Feature Extraction* step and before the *Anomaly Detection* step. In the *Noise Handling* step, a controlled experiment was conducted. Controlled experiments are very common in the area of software engineering today[57]. In the experiment, the independent variables in the groups are different percentages of noise removal. Based on the normal study of researchers [16], these treatments, 2.5%, 5%, 15%, 25% and 50%, are selected. Also, the data set would be too little for training the anomaly detection model if removing more than half of the data set. This is the reason for not choosing more than 50% of the treatment percentage. The dependent variables are Precision, Recall, and F1-score, described minutely in Section 3.6.

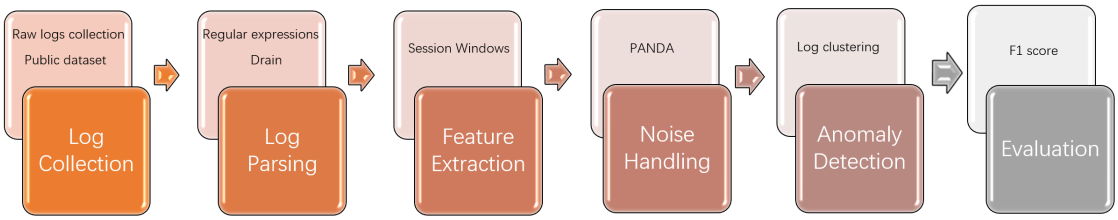

Figure 3.6: Anomaly Detection Process in Group 2

The setup for the *Anomaly Detection* step is exactly the same for both groups. In the *Anomaly Detection* step, two controlled experimental groups(G and B) were designed. Group G is the group with well-tuned parameters of the anomaly detection

algorithm. While group B is the group with badly tuned parameters of the anomaly detection algorithm. In both group G and group B, the same treatments of noise handling were selected.

During the *Noise Handling* and *Anomaly Detection* step, both the Volvo data set and the HDFS data set were used. When it came to the last step, *Evaluation*, only the HDFS data set was applied since the Volvo data is unlabeled. The purpose of training the model with the Volvo data is to show to the company that the anomaly detection model and PANDA can be applied to the production environment. In the end, to evaluate the effect of attribute noise on model learning performance, the results of group G in group one and group two were compared with each other. Similarly, the results of group B of the two groups were compared.

3.3 Setting Up the Tool Chain

3.3.1 Log collection

What is well-known is that logging is necessary for both individuals and systems. For individuals, journaling can help record wonderful or important moments. For systems, event logs record events that occur during the execution of the system to provide a trace that can be used to understand the activity of the system and diagnose problems. When the amount of log data is large and complex, collecting useful log data as the first step in anomaly detection is critical for subsequent log analysis.

3.3.1.1 Collecting Volvo log data

In this thesis study, the log collection process for the Volvo data set is proceeded according to the following steps, as shown in Figure 3.7. This part of the process is the most time-consuming compared to other phases, since this is the starting point of the whole thesis project and everything is a bit unclear.

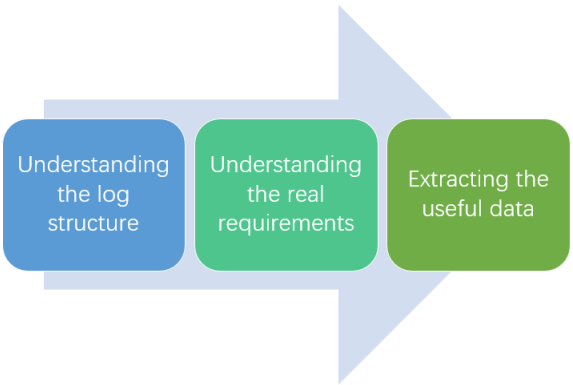

Figure 3.7: Process of Log Collection

- **Understanding the log structure**

  As the first step of the log collection process, several online meetings were held with the developers to help us understand the data structure. Each attribute in the log data of each server was introduced during the meetings.

- **Understanding the real requirements**

  After frequent meetings with colleagues of different responsibilities, such as the manager and solution engineers, three cases of anomalies are defined in the context of the company:

  1. Logs that are tagged as errors but are not errors actually.
  2. Logs that are tagged as errors and they are errors actually.
  3. Logs that are tagged as normal but are errors actually.

  An error log is a record of critical errors encountered in the running of an application, operating system, or server[58]. Typically, there are a number of reasons for an error log, including running out of memory, not being able to connect to the server, and so on. Yet not every error showing up in the logs is one that requires the developer's time to intervene. Sometimes, the failure to connect to the server may just be due to a network reason, which has nothing to do with the server itself. This kind of log belongs to case 1. It would not be worthwhile for engineers to spend a lot of time finding specific log locations and figuring out the cause of those errors. Therefore, in this study, case 2 is focused on. Anomalies in the Volvo data refer to the "real" errors. The real requirement lies in finding those errors in the error logs that really matter.

- **Extracting the useful data**

  To scale down the size of the data, logs generated within one month from one server were used. The server is running in the industry environment. Since the real requirement is to detect "real" errors among those error logs, only error logs in the log files were filtered out as training data. Finally, the data set that consists of error logs generated by one server over a period of one month was collected.

3.3.1.2 Collecting public log data

To further evaluate the results in this study, a public data set HDFS is selected. There are two reasons for selecting this open-source data set: 1. This data set is manually labeled by the original authors. The labels can be used to evaluate the result of the algorithm. While the Volvo data set is unlabeled. 2. The logs in the data set are also chain event logs as the Volvo data set. The identifier is block id. It is similar to the correlation id in the Volvo data set.

This public data set is downloaded from Log Hub[59]. The data set includes two files. As described in 3.1.2, one file is about the raw log data. Another file is about the labels for the log data.

3.3.2 Log Parsing

To easily filter and analyze valuable log information, logs needed to be parsed. A piece of valuable log information can be log messages and server id that are helpful for identifying the problems. The selected tool for parsing the logs is named Drain, mainly due to its stable accuracy and robustness compared with 13 log parsers across 16 real-world log data [25]. The log parsing methods for the two data sets are similar but slightly different. The methods are similar since the data sets are both parsed using the same tool Drain. The parsing methods are slightly different, because the format of the Volvo log data should be converted before applying Drain. While Drain can be applied directly to the public data set.

3.3.2.1 Parsing the Volvo log data

As mentioned in 3.1.1, the format of the raw data is *xml*. In order to make the logs more readable for users and easier to process by Drain later, the Python module *xmltodict* [60] is used to convert the *xml* to the format of *dictionary*. Then the attributes and their corresponding values in the *dictionary* can be accessed. For example, one of the attributes is "Timestamp" and the values are the exact date and time when the logs are generated. Here follows a small example of converting one log event from Figure 3.2.

```
[('Timestamp', '2021-02-23T15:01:18+07:00'), ('Category', 'System'),
('AppDomain', 'Product'), ('Severity', 'Error'),
('Message', 'Message Message2'),
('CorrelationId', '2wwwwwww-a333-4000-8afa-999999e299f5'),
('RequestingNodeId', 'company.server@ASDFGHKKL000200')]
```

Since only the values in the dictionary need to be analyzed, the values in the dictionary are extracted, as shown below.

```
2021-02-23T15:01:18+07:00 System Product Error Message Message2
2wwwwwww-a333-4000-8afa-999999e299f5 company.server@ASDFGHKKL000200
2021-02-23T15:01:23+07:00 System Product Error Message Message4
2wwwwwww-a333-4000-8afa-999999e299f5 company.server@QWERTYUIO000199
```

A regular expression is a logical formula used to manipulate strings. It uses predefined characters and the combination of these characters to form a "regular string". The "regular string" is used to filter strings. Regular expressions are often used to retrieve and replace text that matches a certain pattern. To reduce the latency of parsing logs, needed log content should be extracted and unnecessary punctuation should be removed. With this purpose in mind, regular expressions and Python's

functions are used. The conversions made here are: removing all line breaks and dash symbols in each log, converting all letters to lowercase to ensure consistency (e.g. "Hello" and "hello" are different words for the algorithm) and replacing common punctuation marks (such as : , .) with spaces. The detailed regular expressions and functions examples are shown in Table 3.1. With the help of these expressions, the vocabulary is simplified. Then Drain is applied to this data set. The method is the same as 3.3.2.2.

Regular expressions or functions	Meaning
r'<Event[\s\S]*?</Event>'	Extract log content
r'\n'	Remove line break
'[' + string.punctuation + ']+'	Replace punctuation marks
lower()	Convert letters

Table 3.1: Regular expressions and Functions

3.3.2.2 Parsing public log data

In general terms, an unstructured log contains different system run-time information and free-style raw messages that developers write. The raw log messages are unstructured. An online log parsing tool named Drain is applied to accelerate the parsing process [31] and make the logs easier for the anomaly detection algorithm to process. This tool leverages the principle of clustering to put logs into different log groups. There are plenty of logs that are different in small details but generally similar. One example is that in some logs, the words of the error messages are the same, only some numbers like IP address are different. To limit the number of log groups against which the raw log message needs to be compared, these kinds of logs are grouped together and they correspond to the same event template. In these cases, regular expressions are adopted again to further simplify the data set. Figure 3.8 shows an example below. In the example, two event templates are generated. Log message 1 and 2 belong to event 1. Similarly, log message 3 and 4 belong to event 2. During this step, only the numbers in the messages are simplified, so this simplification does not affect the performance of anomaly detection later.

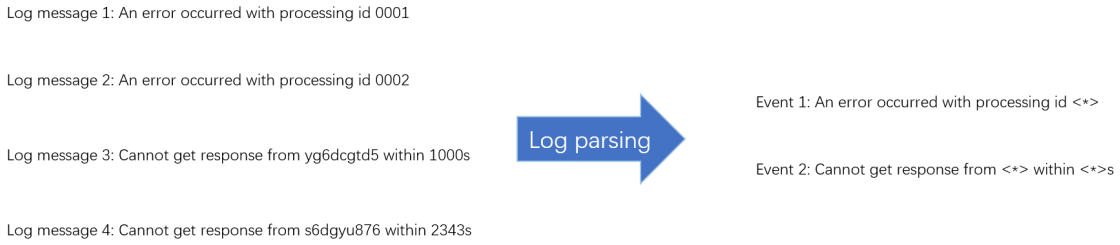

Figure 3.8: Log Parsing Results

Two CSV files are generated after the logs are parsed. In the files, one column corresponds to one attribute. One file stores the event templates that include *EventId*,

EventTemplate, and *Occurrences* as shown in Figure 3.9. Each event template has a unique Id(E1, E2,...). *Occureences* means the number of times the messages corresponding to these templates appear in the data set. Another file stores structured data with some extra columns such as *ParameterList* and *LineId*. An instance can be seen in Figure 3.10. *LineId* indicates the row in which the log data is located. *ParameterList* refers to how many bits of parameters(numbers or letters) are replaced with '<*>'. The event templates will be used as the input for the next step, feature extraction.

EventId	EventTemplate	Occurrences
E1	An error occurred with processing id <*>	590
E2	Cannot get response from <*> within <*>s	17

Figure 3.9: Event Templates

LineId	TimeStamp	Message	EventId	EventTemplate	ParameterList
1	20220422T00:00:01Z	An error occurred with processing id 0001	E1	An error occurred with processing id <*>	['4']
2	20220423T00:07:02Z	Cannot get response from yg6dcgtd5 within 1000s	E2	Cannot get response from <*> within <*>s	['9','4']

Figure 3.10: Structured Data

3.3.3 Feature Extraction

After the logs are parsed, the event templates should be encoded into numerical feature vectors to make preparations for applying machine learning models. This step is critical, since it can greatly affect the quality of the model. The model can be overfitted when the dimensionality of the features is too high [61]. Also, too many features can cost much computational time [61].

A grouping technique, session windows, is adopted to optimize the quality of features. The input is the parsed log events and the output is an event count matrix. In the HDFS data set, each log event has a specific block id. The block ids are used as the *partition key* of the session windows. The log events that have the same block id all convey the relevant information about the block. Thus, they will be grouped together in the same window. In other words, each window has a unique block id. Figure 3.11 gives a small example. Suppose from E1 to E5 are five log event templates and each of them has a block id(Bid). Each log event template also has a unique event id(E1, E2,...) generated from the last step 3.3.2. The log events with Bid 123 are grouped together in session window 1. Similarly, other log events with Bid 456 are grouped together in session window 2. Then each window is converted into an event count vector. For example, E1 occurs twice in session window 1, so it will be counted as 2 in the event count vector. Likewise, E2 is counted as 0 in the vector owing to that it never occurs in session window 1. Those vectors together form an event count matrix as the output.

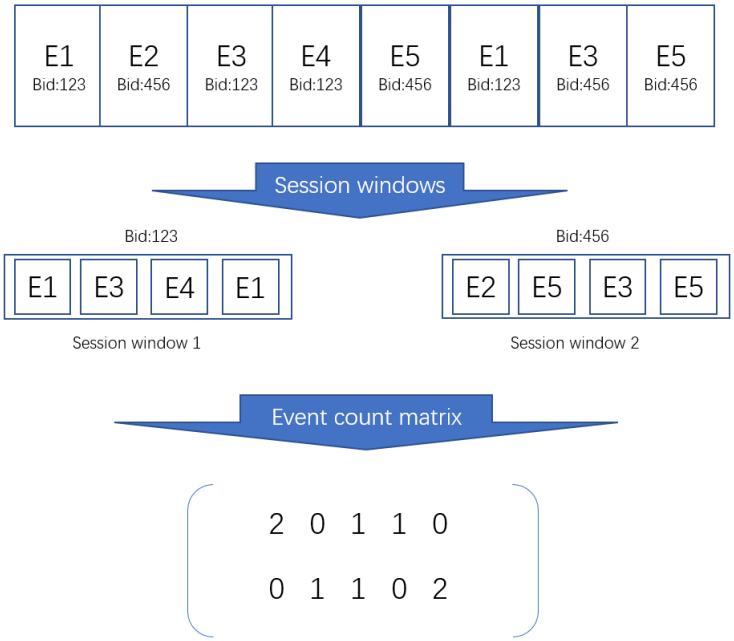

Figure 3.11: Session Windows Example

3.3.3.1 TF-IDF

TF-IDF is commonly used for information retrieval and text mining [62, 63]. In our context of anomaly detection in application logs, TF means the frequency that each log event appears in the log sequence. The results of TF calculation can be found in the event count matrix that is generated in Figure 3.11.

$$IDF = \log(\frac{The\ total\ number\ of\ log\ sequences}{(The\ number\ of\ log\ sequences\ containing\ the\ log\ event) + 1}) \quad (3.1)$$

IDF refers to how many log sequences containing this log event account for the total number of log sequences. To calculate IDF, the formula 3.1 is applied. Figure 3.12 exemplifies the process of calculating TF-IDF. In the figure, we assume there are two log sequences altogether. These log sequences contain log events that have five different log event ids(E1, E2, E3, E4, and E5). The number of log sequences that contain E1 is 1. Consequently, the value of IDF for E1 is $log(2/(1+1)) = 0$. The IDF values of the rest log events are calculated in the same manner. After the value of IDF for each log event is calculated, the values will be formed as one line $(0\ 0\ -0.18\ 0\ 0)$ that contains IDF values for every log event. The line will be repeatedly put into a matrix. The number of line repetitions equals to the total number of log sequences. In this example, the line is repeated twice. Next, the values of TF-IDF can be calculated by multiplying the values in the event count matrix and IDF matrix.

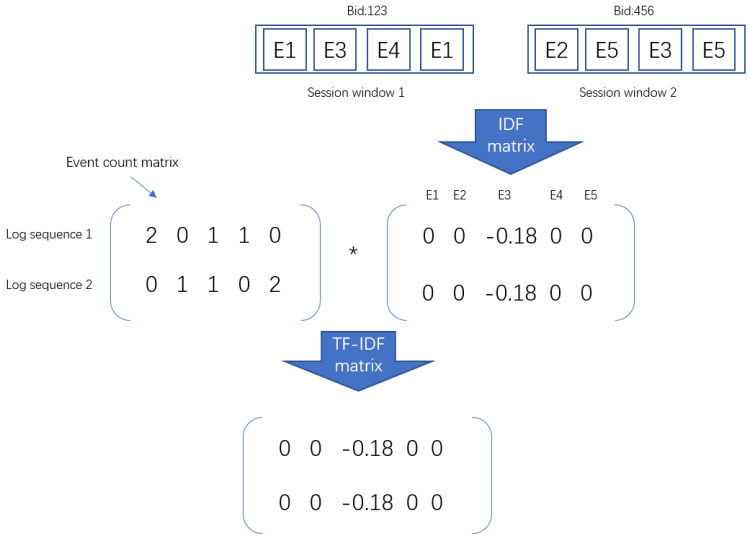

Figure 3.12: TF-IDF Calculation Example

3.4 Noise Handling

This study focus on examining the effect of handling instances with attribute noise on the learning performance for anomaly detection in application logs. The selected attribute noise removal approach is PANDA [1, 16] due to its reliability validated by a software engineering expert. PANDA is also the first attribute noise handling algorithm that can be run without the need of class labeled data [1]. The input is a matrix that is obtained from processing the event count matrix by TF-IDF in *Feature Extraction* step. Each line in the matrix is an instance, also called a log sequence. Each log event template is considered an attribute. We set the value of bin size to the total number of log sequences, which means all the instances of one attribute will be put in the same bin. The reason is that partitioning the instances into different bins will have the risk of introducing additional noise in the data set [1]. Also, there is no interest to investigate the impact of different bin sizes on anomaly detection in this study.

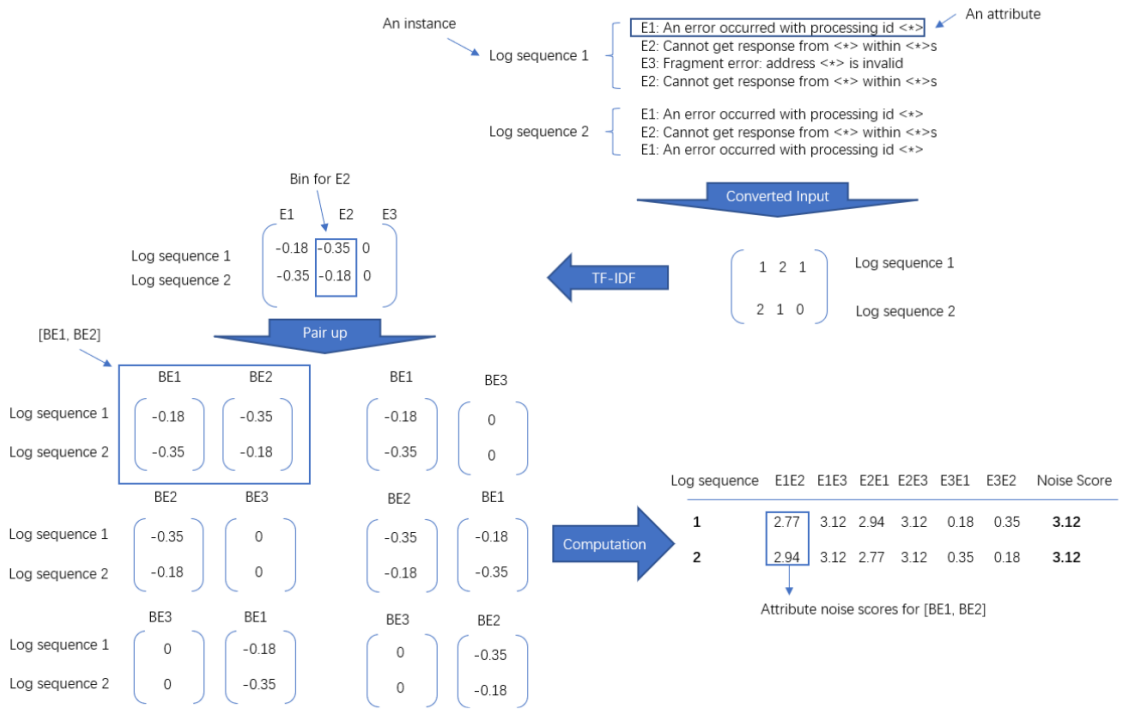

Figure 3.13: Noise Handling Example

Figure 3.13 gives an example. Assume there are two log sequences. Each attribute has a score computed by applying TF-IDF in each log sequence. According to the set bin size, all of the instance values for the same attribute will be put in the same bin. The bin for E2(BE2) in the figure is an example of one bin. Then each bin of attributes would pair up with other bins of attributes. The pairs are ordered. For example, [BE1, BE2] and [BE2, BE1] are different pairs. In each pair, the conditional mean and standard deviation of the first attribute are computed. For instance, for the pair [BE1, BE2], the bin of the first attribute is BE1. The mean of BE1 is $(-0.18 + (-0.35))/2 = -0.265$. The standard deviation of BE1 is $\sqrt{\frac{(-0.18-(-0.265))^2+(-0.35-(-0.265))^2}{2-1}} = 0.085$. Then the discretized values of the bin of the second attribute are computed. The calculations are: $(-0.35) - (-0.265/0.085) = 2.77$ and $(-0.18) - (-0.265/0.085) = 2.94$. These scores refer to the attribute noise scores of the pair [BE1, BE2]. Since the scores should be greater than 0, their absolute values are taken which are also 2.77 and 2.94. The computation is repeated in every pair. The output in this step is an ordered list with the log sequences from the highest noise possibility to the lowest possibility. The final noise score for each log sequence is the max score in each line. Those scores before the last score are the attribute noise scores for each pair of attribute bins such as "E1E2" stands for the noise score for the pair [E1,E2] In the example, these two log sequences are of the same highest possibility to be noise in the data set.

With this ordered list, different percentages of noise data could be removed. In the experiment, the percentage of noise removal in each of the four experimental groups is 2.5%, 5%, 15%, 25%, and 50%. Different percentage of noise removal means removing different percentages of log sequences. The noise data would be

removed starting with the log sequence that is of the highest score in the list. For example, removing 5% noise data refers to removing 5% log sequences that of highest possibility to be noise in the data set.

3.5 Anomaly Detection

After the data was preprocessed, an anomaly detection algorithm log clustering was run. There are several reasons for choosing the log clustering algorithm. First, it is developed to especially ease identify log-based problems. Second, it is more reliable than other approaches such as the approach proposed by Shang et al.[64] and keyword searching. The precision and accuracy of this algorithm are much higher [20]. Also, it has already been successfully applied to the real industry environment [20].

In this log clustering algorithm, two parameters are given different values to get different evaluation results of the model. The two parameters that the algorithm allows to configure are the *max distance* and *distance threshold*. Then, one model with good parameters and one with bad parameters are the output in this step. The process of log clustering can be seen in Figure 3.14.

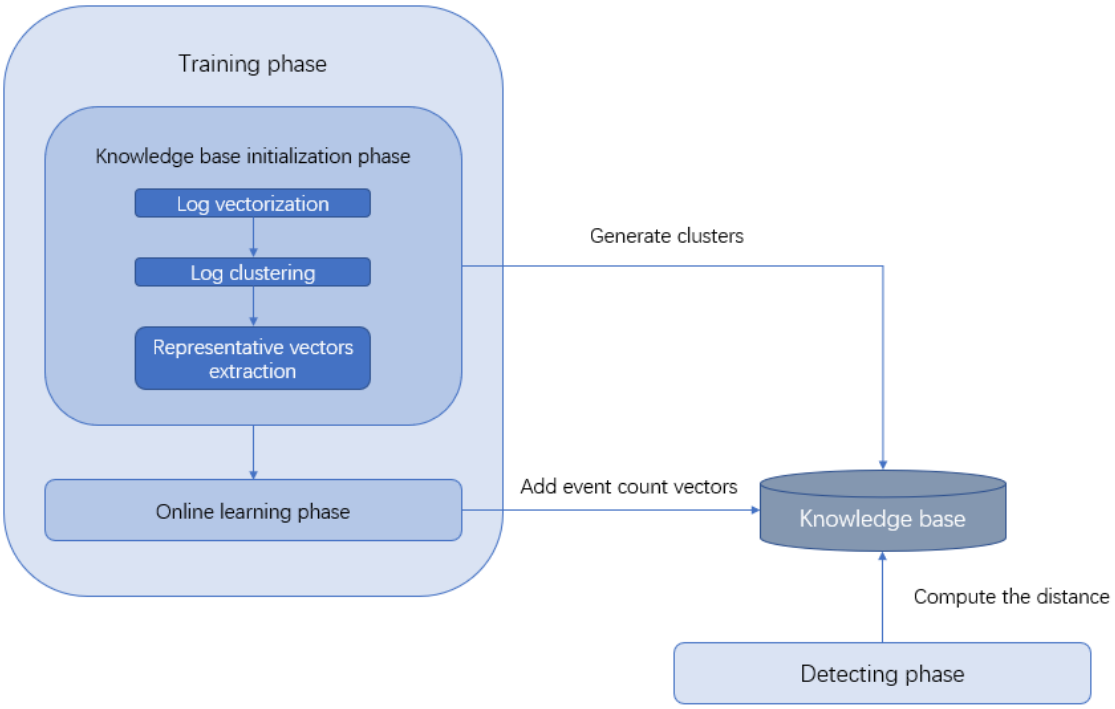

Figure 3.14: Processes of Log Clustering

The training phase of log clustering is mainly run in two phases: the knowledge base initialization and online learning phases [20]. In the knowledge base initialization phase, there are three steps: log vectorization, log clustering, and representative vector extraction. Part of the training instances are trained during this phase. In the first step, log sequences are vectorized into an event count matrix by using TF-IDF. After vectorization, the vectors are clustered together into two clusters as the

knowledge base, one normal and one abnormal. Agglomerative Hierarchical clustering technique [22] is utilized in this step. Then, in each cluster, a representative vector is selected based on the sequence's average distance to other log sequences in the same cluster.

The other part of the training instances are trained in the online learning phase. In this phase, the distances from the newly added event count vectors to the representative vectors are compared to the distance threshold. If the minimum distance is less than the threshold, the cluster that is closest to the new vector will add this vector and update the representative vector in the cluster. Otherwise, the newly added event count vector will be used to create a new cluster.

In the training phase, three parameters are given different values to get different results for the model. They are *maximum distance*, *distance threshold*, and *init samples*. *Maximum distance* means the maximum distance between the log sequences in the same cluster. *Distance threshold* is a threshold that decides when to stop clustering. As Section 2.1.1.1 described, if the maximum distance between two clusters exceeds the threshold, the clustering process will be terminated. *Init samples* refers to the number of training samples trained in the knowledge base initialization phase. For instance, if the total number of training samples is 3000 and the *init samples* is set to 1000, then there will be 2000 training samples used in the online learning phase. Then two models with good and bad parameter values are then selected to detect anomalies. Similar to the online learning phase, the distance of the new log sequence is computed to the representative vectors in the knowledge base during the detecting phase. The new log sequence will be reported as an anomaly when the minimum distance is larger than the threshold. In other cases, the nearest cluster to the log sequence will add the sequence to the cluster, which can be normal or abnormal.

3.6 Evaluation

To evaluate the performance of the model, the precision, recall, F1-Score, and MCC of the model were measured. F1-Score combines precision and recall. MCC was selected because it is still reliable even when the data is unbalanced [53]. In this step, only the HDFS data set was applied. The reason why the Volvo data set was not used is that the Volvo data set is unlabeled. After the anomaly detection model ran in different settings, the file (as shown in Figure 3.4) that stores the BIDs and corresponding labels was used to calculate the *true positive*, *false positive*, *true negative* and *false negative*. The functions that are utilized to do the calculations are $sklearn.metrics.precision\_recall\_fscore\_support$ and $sklearn.metrics.matthews\_corrcoef$[65]. The detailed results of group one(without noise handling) and group two(with noise handling) will be shown in Section 4. We will compare the results of these two groups. Besides, we will make a comparison between the results of good model quality and bad model quality in different scenarios of attribute noise removal.

3.7 Interviews

The interview in this study is semi-structured. Semi-structured interviews are flexible conversations in which the topics and order of the interview questions can be predetermined or can be adapted flexibly during the interviews [66]. The conversations are free to vary between participants. This can happen because there are constantly new ideas arising during the conversations. Semi-structured interview has proven to be a popular method of data collection because it is both versatile and flexible [67]. It has already been employed in different research areas, such as research on group development and group maturity when building agile teams[68].

The interviews aim to understand how the anomaly detection process actually works and what the users expect from the anomaly detection tool from the industry's perspective. In total six people were interviewed and the interviews were all conducted via online video calls, since the interviewees are living in different countries. To keep anonymity, the interviewees will be named I1, I2, I3, I4, I5, and I6. The interviewees come from two cooperative teams in the same department and they had all been working with the application logs in some way. In order to know a bit about the background of the interviewees and understand how they troubleshoot the errors in the logs manually, the interview questions are designed as below:

- What's your area of expertise?
- How long have you been working in this company?
- How often would you look at the application logs?
- How much time do you usually spend in a month on querying logs and solving problems? How much time and energy do you think it will take?
- What's your process of solving the problem manually?
- How harmful is it if the anomaly detection tool marks cases that are no anomalies? When is it too much? Is there too much? What is the threshold?
- How harmful is it if the anomaly detection tool misses anomalies? When is it too much? Is there too much? What is the threshold?
- What would make you stop using the tool? In which scenario that you would stop trusting the tool?

Table 3.2 specifies the background of the interviewees in this study, including how long the interviewees have been working for the company, their role on the team and how often they check the application logs.

ID	Experience	Role	Frequency of Checking Logs
I1	9 years	Business analyst	Daily
I2	7 months	Support analyst	When getting a case
I3	22 years	Solution leader	Whenever in duty
I4	12 years	Software engineer	Whenever in duty
I5	12 years	Delivery leader	Daily
I6	12 years	Delivery manager	When needing extra help

Table 3.2: Background of Interviewees in this Study

4
Results and Discussions

In this section, we analyze the results in terms of configurations for anomaly detection. In addition to this, we compare the results from training the model with data that is (1) without noise handling, and (2) with different percentages of attribute noise removal. We report the results by evaluating precision, recall, F1-score, and MCC. Furthermore, we discuss the results of the interviews. Finally, we reflect on some limitations and delimitations that this study has.

4.1 Anomaly Detection

In the following subsections, we present and compare the results from training the anomaly detection model with good and bad-tuned parameters. Also, we evaluate the results in the group that experimented with noise handling and another that is without noise handling. Next, we answer the research question of this study.

4.1.1 Baseline measures for anomaly detection: good and bad configurations

In order to evaluate how much noise handling can improve the process of anomaly detection, PANDA is applied in situations with different degrees of anomaly detection model quality. The motivation for this is to check whether the results are robust against differences in model quality. Good quality of the model in this study is crucial, so there is a need to predict correct noise data and then remove them. Otherwise, it would still cause lots of time and energy to manually check and troubleshoot errors.

Question: To gain a baseline that is representative when comparing anomaly detection with noise handling to that without noise handling, what configuration should be used for anomaly detection?

To gain robust results, two different configurations were decided to be used. One configuration is the anomaly detection model with good parameters. Another is the anomaly detection model with bad parameters. Thus, only the results of the model that is trained with the HDFS data set are shown here. Researchers have empirically set the value of *distance threshold* in their experiments to 0.5 [20]. So we applied different values of *distance threshold* that centered around 0.5. The range of the explored values is from 0.1 to 1.9. Similar to *distance threshold*, the explored values

of *max distance* also range from 0.1 to 1.9. The following Figure 4.1 and Figure 4.2 show different evaluation scores for different parameter values. When measuring the baseline, all the scores are obtained by training the model without noise handling. The evaluation scores include precision, recall, and F1 score. In Figure 4.1, there is a turning point when the value equals to 1.0. When the value is larger than 1.0, F1 score and precision almost score 0, while recall almost scores 1.0. It can be seen obviously that the model would be of better quality when the value of Figure 4.1 is less than 1.0. According to Figure 4.2, the scores of precision, recall, and F1 score all dropped sharply. Thus, the values of *distance threshold* that are less than 1.0 should be selected for Group G.

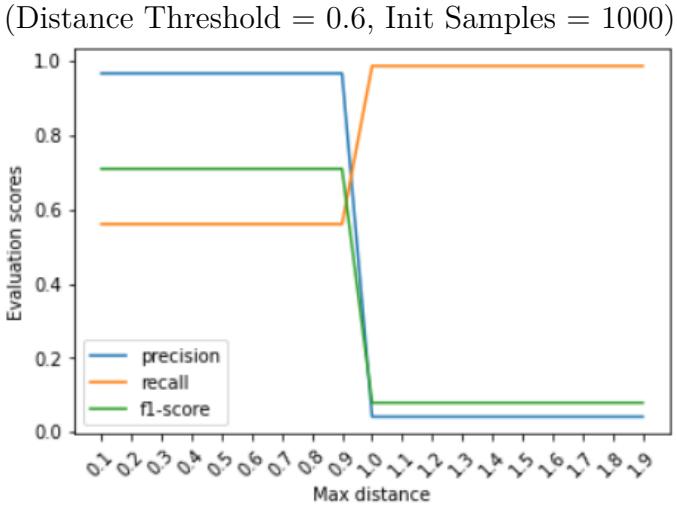

Figure 4.1: Evaluation Scores of Max Distance

Also, multiple values of *init samples* are tried during the training phase as shown in Figure 4.3. These training samples are trained in the knowledge base initialization phase that is offline. The total number of training samples is 3969. Those samples should be trained either in the knowledge base initialization phase or the online learning phase. In Figure 4.3, we applied the values from 0 to 3800. Clearly, the evaluation scores keep the same when values of *init samples* are changing from 0 to 3800. This means that the number of training samples used in the knowledge base initialization phase would not affect the quality of the model.

When deciding values for Group G, a value, 0.6, that is less than 1.0 is chosen for both *distance threshold* and *max distance*. When deciding values for Group B, a value, 1.5, that is larger than 1.0 is chosen for *max distance*. Yet the value of *distance threshold* is still 0.6. When experimenting with the combination of the values of these two parameters, we also evaluated another two combinations. They are: *distance threshold*: 1.5, *max distance*: 1.5; *distance threshold*: 1.5, *max distance*: 0.6. In these two combinations, the *distance threshold* values are both 1.5. However, the scores of precision, recall, and F1-score are all 0 for these two combinations. It is of no research value to continue investigating these two combinations. Thus, the value for *distance threshold* was decided to be 0.6. Hence, as a result, the values of group

G and group B are summarized in Table 4.1.

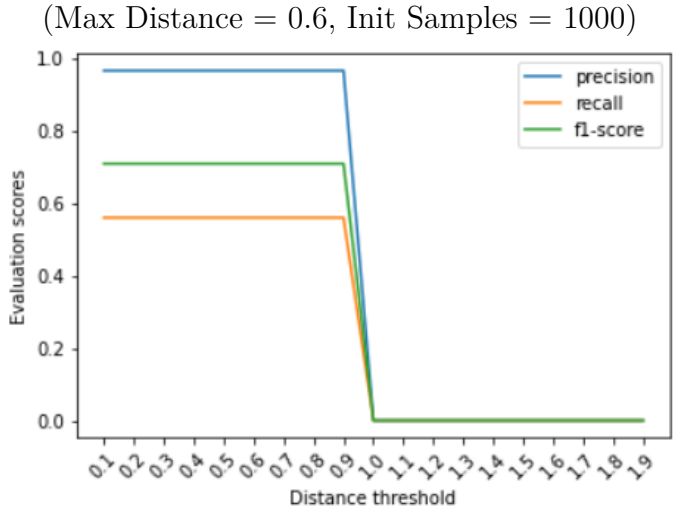

Figure 4.2: Evaluation Scores of Distance Threshold

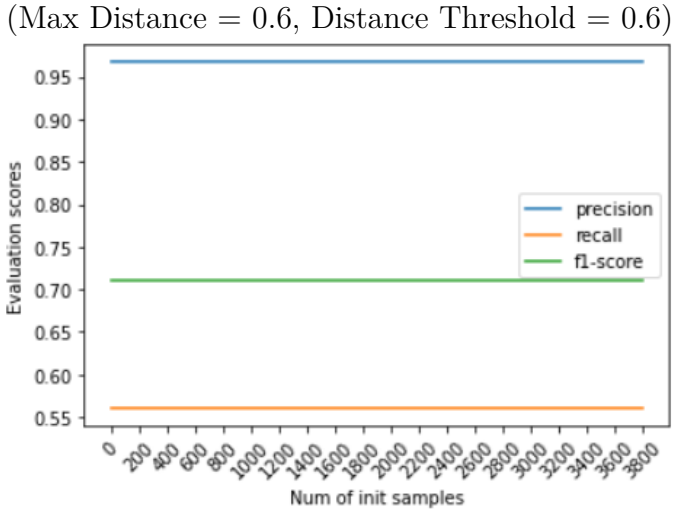

Figure 4.3: Evaluation Scores of Init Samples

Experimental Group	Max Distance	Distance Threshold	Init Samples
Group G	0.6	0.6	1000
Group B	1.5	0.6	1000

Table 4.1: Values of Parameters in Group G and Group B

4.1.2 Group One: Anomaly Detection without Handling Attribute Noise

As described in Section 3.2, the results of group one are used as a baseline for comparing the effect of handling different percentages of attribute noise on anomaly

detection. The process of group one is shown in Figure 3.5. During the *anomaly detection* step, the anomaly detection model was trained with two groups of parameter values that are generated in Group G and Group B respectively. The results are shown in Table 4.2. The values in the results are the average values of 30 runs of the program.

Experimental Group	Precision	Recall	F1 Score	MCC
Group G	0.967	0.561	0.710	0.729
Group B	0.041	0.987	0.079	0.032

Table 4.2: Results in Group One

4.1.3 Group Two: Anomaly Detection with Handling Attribute Noise

The process of anomaly detection with attribute noise handling can be seen in Figure 3.6. Before training the anomaly detection model, the attribute noise handling method PANDA was applied. Then during *anomaly detection* step, in both group G and group B, the model was trained with preprocessed data that has had different percentages of noise removed. The results are summarized in Table 4.3. In every experimental group, the results are the average values obtained after running the model for 10 times in each scenario of different noise removal, respectively.

Experimental Group	Percentage of Noise Removal	Precision	Recall	F1 Score	MCC
Group G	2.5%	0.967	0.558	0.707	0.727
	5%	0.965	0.557	0.706	0.726
	15%	0.975	0.552	0.705	0.726
	25%	0.988	0.552	0.709	0.730
	50%	1	0.552	0.712	0.731
Group B	2.5%	0.042	0.987	0.080	0.032
	5%	0.041	0.987	0.080	0.031
	15%	0.044	0.986	0.084	0.032
	25%	0.050	0.986	0.094	0.034
	50%	0.075	0.986	0.140	0.050

Table 4.3: Results in Group Two

RQ: How can we improve the predictive performance of a learner for anomaly detection in application logs by handling attribute noise?

To answer this research question, we compared the results of anomaly detection with with different percentages of noise removal to that without noise handling. The results are evaluated by examining the precision, recall, f1-score, and MCC in Table 4.2 and 4.3. From these two tables, we can see that the values of precision,

f1-score, and MCC continue to increase slightly as the percentage of noise removal increases. Although the values of them go down a little bit when removing 5% of noise data, the overall trend of the results remains increasing. The values of recall continue to decrease slightly as the percentage of noise removal increases. Also, it can be observed that the change in the values of precision is greater than the change in the values of recall. The overall change might seem small. Nevertheless, there might be a real difference with large log data. Since the results keep the same trend of increasing or decreasing, we can deem that the results for both experimental groups are robust against the quality of the anomaly detection model. With changing quality of the model, we can expect the results to stay stable. Thus, the results reveal that PANDA can help improve the precision, F1-score, and MCC of the anomaly detection model, with the cost of a slightly reduced recall.

4.2 Interviews

Six colleagues from two teams in the company were interviewed to have a better understanding of how anomaly detection works in the company and the company's expectations of the anomaly detection tool's quality. They were all interviewed through online video calls. Also, to facilitate the summary of the interview, all conversations were recorded.

During the interviews, in total 8 questions were asked to the interviewees. For the first three questions, the answers are summarized in Table 3.2. For the rest of the questions, the answers are as below:

- How much time do you usually spend in a month on querying logs and solving problems? How much time and energy do you think it will take?

  Some colleagues would spend much time looking at the logs and solving the problems, for example, more than 50% working time. While other colleagues would only check the logs when they are on duty(one day per month) and something serious happens. Some of them felt it can take plenty of time but not energy. While other participants said that it could be very tiring as well. They all considered it will be much better if the process of querying logs and finding problems could be more efficient.

- What's your process of solving the problem manually?

  According to the interviewees' answers, the process of solving every problem can be different. But in general, the process can be deemed to: first, look at the problem and find the error logs' location according to the timestamp of the logs. Second, go into the details of the error logs and try to understand what they mean. Then they would ask the customer for more information sometimes, for instance, what operations they have done. The majority of problems would be solved by themselves, but it can be time-consuming to communicate with other related staff.

- How harmful is it if the anomaly detection tool marks cases that are no anomalies? When is it too much? Is there too much? What is the threshold?

  Few participants thought it could be very annoying if the normal logs are detected as anomalies. Others said it would be okay if the tool reports fewer than 50% false alarms. One participant suggested it could be nice if the anomalies that are detected by the tool had different levels. For example, "red" indicates "supercritical" and "yellow" means "be aware".

- How harmful is it if the anomaly detection tool misses anomalies? When is it too much? Is there too much? What is the threshold?

  All the participants considered this situation is more critical than the situation mentioned in the last question. Some of them thought it could be extremely harmful if the anomalies that should have been detected were not detected. However, others felt fine if the tool can be still helpful. Speaking of the threshold, one of the participants expected the tool can detect more than 80% true anomalies. Half of them thought it is better than having no such tool.

- What would make you stop using the tool? In which scenario that you would stop trusting the tool?

  More than half of the participants would stop trusting the tool if it reports more than 50% false cases. One participant mentioned that maintainability is critical for the tool. If the users are unable to maintain the tool(e.g. modifying the code of the tool), they would stop using it. Another participant held the opinion that if the tool is not easy to set up and learn, the tool will be "discarded".

4.2.1 Discussion of Results in Context of Interviews

According to the interview, our anomaly detection tool could be helpful when speaking of the process of solving the problem. With the anomaly detection tool, the users don't need to spend much time finding the error logs' location. Instead, the tool would detect the error logs as anomalies and report them to the users. Thus, using this tool will make the problem-solving process more efficient.

One participant mentioned the expectation of the tool was that the tool could detect more than 80% of the exceptions. This means that the value of recall should be more than 80%. Whereas, we can see from Table 4.2 and 4.3, the values of recall are all under 80% in Group G. Also, although the values of recall are more than 80% in Group B, the values of precision are low. Therefore, the tool does not meet this requirement currently and the improvements gained by attribute noise handling cannot change the situation.

When it comes to the scenarios in which users would distrust the tool, the participants discussed that they would stop using the tool when more than 50% false cases are reported by the tool. Connecting this scenario to the results, it means that the value of precision should be greater than 50%. As presented in Table 4.2 and 4.3, the values of precision are all far more than 50% in Group G. Besides this, there is a trend of increase for the values of precision as the percentage of noise removal increases. Although in none of our cases the noise handling made the quality cross these thresholds, this anomaly detection tool can be beneficial to users when it comes to trust in the tool.

4.3 Limitations and delimitations

Limitations described here are divided into threats to external and internal validity:

- **Threats to External Validity**

  External validity refers to the extent to which the results of our research can be applied to real software applications [69].

  Source data. The data that is used for evaluation in this study only comes from one distributed file system. However, this data set can be representative, since the structure of this data set is common. The main information that needs to be processed by this anomaly detection tool, such as log messages, is contained in almost every data set that is generated from different application systems.

- **Threats to Internal Validity**

  Internal validity considers the degree that whether the reported causal effect in our study is trustworthy [70].

  Anomaly detection model. The algorithm that is selected to train the anomaly detection model is only - Log Clustering. This is a common algorithm. The performance of this anomaly detection algorithm might be different from other algorithms. However, the purpose of this study is to research how to improve the predictive performance of the anomaly detection model by handling attribute noise, so whether the model itself and the parameters are the best selected is not the focus. Thus, we did not spend extensive time experimenting with other models. This limitation will be in analysis in future work.

  Information extraction. During the procedure of data collection, some important information may be missed, which would have a negative impact on the evaluation results. For example, the method of feature extraction is session window. The logs are grouped into session windows based on *partition key*. In the Volvo data set, only log data with correlation id was collected. In the

HDFS data set, only log data with block id was collected. Logs that are without correlation id may also contain important information. This threat was mitigated by checking the results of the training model.

Configuration. During the noise handling step, the bin size was set to be equal to the total number of log sequences. In this case, all the instances of one attribute are put in one bin. The bin size may have an effect on noise ranking. Due to time limitations, We did not delve into this. We will address this in future work.

One delimitation in this study is that only the effect of attribute noise handling is investigated. Since the company's data is unlabeled, there was no interest to research the effect of class noise in this case.

5
Conclusion and future work

This study provides an experimental analysis of handling attribute noise along with a model that automatically detects anomalies. We compare the results of experimenting on (1) the original and (2) attribute-noise cleaned data with different percentages of noise removal by examining the precision, recall, F-score, and MCC. The results suggest that the noise handling approach, PANDA, improves the precision, F1-score, and MCC of the anomaly detection model in application logs. Additionally, this study is practical since the anomaly detection tool can be applied in different systems.

There is still some future work that needs to be done. One is that some other attribute noise elimination methods can be adopted to see if they will enhance performance more effectively than PANDA. Also, different bin sizes of PANDA can be tuned to check if the bin size has a great effect on noise ranking. Then, other data sets and anomaly detection algorithms could be used to investigate whether this noise-eliminating approach can also improve performance to the same extent. Another work is some class-noise handling approaches can be researched to examine the effect of class noise on anomaly detection in application logs.

Bibliography

[1] J. D. Van Hulse, T. M. Khoshgoftaar, and H. Huang, "The pairwise attribute noise detection algorithm," *Knowledge and Information Systems*, vol. 11, no. 2, pp. 171–190, Feb 2007, doi: 10.1007/s10115-006-0022-x. [Online]. Available: https://doi.org/10.1007/s10115-006-0022-x

[2] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, p. 67, May 2016, doi: 10.1186/s13634-016-0355-x. [Online]. Available: https://doi.org/10.1186/s13634-016-0355-x

[3] H. Zawawy, K. Kontogiannis, and J. Mylopoulos, "Log filtering and interpretation for root cause analysis," in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–5, doi: 10.1109/ICSM.2010.5609556. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5609556

[4] A. Grover, "Anomaly detection for application log data," 2018, MSc Dissertation, Dept.of Comput. Sci., Univ. of San Jose State, USA. doi: 10.31979/etd.znsb-bw4d. [Online]. Available: https://scholarworks.sjsu.edu/etd_projects/635

[5] M. Ahmed, A. N. Mahmood, and M. R. Islam, "A survey of anomaly detection techniques in financial domain," *Future Generation Computer Systems*, vol. 55, pp. 278–288, 2016, doi: 10.1016/j.future.2015.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X15000023

[6] 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)*, 2016, pp. 207–218, doi: 10.1109/ISSRE.2016.21. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7774521

[7] S. Gupta and A. Gupta, "Dealing with noise problem in machine learning data-sets: A systematic review," *Procedia Computer Science*, vol. 161, pp. 466–474, 2019, doi: 10.1016/j.procs.2019.11.146. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919318575

[8] X. Zhu and X. Wu, "Class noise vs. attribute noise: A quantitative study," *Artificial Intelligence Review*, vol. 22, no. 3, pp. 177–210, Nov 2004, doi: 10.1007/s10462-004-0751-8. [Online]. Available: https://doi.org/10.1007/s10462-004-0751-8

[9] D. Guan, W. Yuan, Y.-K. Lee, and S. Lee, "Identifying mislabeled training data with the aid of unlabeled data," *Applied Intelligence*, vol. 35, no. 3, pp. 345–358, Dec 2011, doi: 10.1007/s10489-010-0225-4. [Online]. Available: https://doi.org/10.1007/s10489-010-0225-4

[10] X. Zhu, X. Wu, and Q. Chen, "Eliminating class noise in large datasets," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03. AAAI Press, 2003, p. 920–927. [Online]. Available: https://dl.acm.org/doi/10.5555/3041838.3041954

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, jul 2009, doi: 10.1145/1541880.1541882. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[12] S. Thudumu, P. Branch, J. Jin, and J. J. Singh, "A comprehensive survey of anomaly detection techniques for high dimensional big data," *Journal of Big Data*, vol. 7, no. 1, p. 42, Jul 2020, doi: 10.1186/s40537-020-00320-x. [Online]. Available: https://doi.org/10.1186/s40537-020-00320-x

[13] D. Wettschereck, "A study of distance-based machine learning algorithms," 1994, PhD Dissertation, Dept. of Comput. Sci., Univ. of Oregon State, Oregon, USA. [Online]. Available: https://ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/zw12z7835

[14] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003, biometrics, doi: 10.1016/S0031-3203(02)00060-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320302000602

[15] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, jul 2017, doi: 10.1145/3068335. [Online]. Available: https://doi.org/10.1145/3068335

[16] K. W. Al-Sabbagh, M. Staron, and R. Hebig, "Improving test case selection by handling class and attribute noise," *Journal of Systems and Software*, vol. 183, p. 111093, 2022, doi: 10.1016/j.jss.2021.111093. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221001904

[17] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 291–301, doi: 10.1145/581339.581377. [Online]. Available: https://doi.org/10.1145/581339.581377

[18] G. K. Baah, A. Gray, and M. J. Harrold, "On-line anomaly detection of deployed software: A statistical machine learning approach," in *Proceedings of the 3rd International Workshop on Software Quality Assurance*, ser. SOQUA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 70–77, doi: 10.1145/1188895.1188911. [Online]. Available: https://doi.org/10.1145/1188895.1188911

[19] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989, doi: 10.1109/5.18626. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/18626

[20] Q. Lin, H. Zhang, J.-G. 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. [Online]. Available: https://ieeexplore.ieee.org/document/7883294

[21] L. Yang *et al.*, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1448–1460, doi: 10.1109/ICSE43902.2021.00130. [Online]. Available: https://ieeexplore.ieee.org/document/9401970

[22] J. C. Gower and G. J. S. Ross, "Minimum spanning trees and single linkage cluster analysis," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54–64, 1969, doi: 10.2307/2346439. [Online]. Available: http://www.jstor.org/stable/2346439

[23] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020, doi: 10.48550/ARXIV.2008.06448. [Online]. Available: https://arxiv.org/abs/2008.06448

[24] T. G. Team, "Log file parsing," graylog.org, https://www.graylog.org/post/log-file-parsing/, Jul. 2020, (accessed Jan. 17, 2023).

[25] J. Zhu *et al.*, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 121–130, doi: 10.1109/ICSE-SEIP.2019.00021. [Online]. Available: https://ieeexplore.ieee.org/document/8804456

[26] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. USA: USENIX Association, 2014, p. 217–231. [Online]. Available: https://dl.acm.org/doi/10.5555/2685048.2685066

[27] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298, doi: 10.1145/3133956.3134015. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[28] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *Proceedings of the*

2010 USENIX Conference on USENIX Annual Technical Conference, ser. USENIXATC'10. USA: USENIX Association, 2010, p. 24. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/1855840.1855864

[29] N. Aussel, Y. Petetin, and S. Chabridon, "Improving performances of log mining for anomaly prediction through nlp-based log parsing," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 237–243, doi: 10.1109/MASCOTS.2018.00031. [Online]. Available: https://doi.org/10.1109/MASCOTS.2018.00031

[30] W. Meng *et al.*, "Logparse: Making log parsing adaptive through word classification," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020, pp. 1–9, doi: 10.1109/ICCCN49398.2020.9209681. [Online]. Available: https://ieeexplore.ieee.org/document/9209681

[31] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40, doi: 10.1109/ICWS.2017.13. [Online]. Available: https://ieeexplore.ieee.org/document/8029742

[32] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 311–322, doi: 10.1145/1066157.1066193. [Online]. Available: https://doi.org/10.1145/1066157.1066193

[33] T. Akidau *et al.*, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endow.*, vol. 8, no. 12, p. 1792–1803, aug 2015, doi: 10.14778/2824032.2824076. [Online]. Available: https://doi.org/10.14778/2824032.2824076

[34] R. Alex, E. Florian, C. Giacomo, H. Jason, M. Mary, and K. Mike, "Session window (azure stream analytics)," learn.microsoft.com, https://learn.microsoft.com/en-us/stream-analytics-query/session-window-azure-stream-analytics, Oct. 2021, (accessed Jan. 17, 2023).

[35] L. Liu and M. T. Özsu, *Encyclopedia of database systems*, 1st ed. Springer New York, NY, 2009, vol. 6, pp. 3037, doi: 10.1007/978-0-387-39940-9. [Online]. Available: https://link.springer.com/referencework/10.1007/978-0-387-39940-9

[36] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "Research-paper recommender systems: a literature survey," *International Journal on Digital Libraries*, vol. 17, no. 4, pp. 305–338, Nov 2016, doi: 10.1007/s00799-015-0156-0. [Online]. Available: https://doi.org/10.1007/s00799-015-0156-0

[37] C. M. Teng, "Combining noise correction with feature selection," in *Data*

Warehousing and Knowledge Discovery, Y. Kambayashi, M. Mohania, and W. Wöß, Eds., vol. 2737. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 340–349, doi: 10.1007/978-3-540-45228-7_34. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-45228-7_34

[38] L. Yu, X. Huang, and H. Yin, "Can machine learning paradigm improve attribute noise problem in credit risk classification?" *International Review of Economics Finance*, vol. 70, pp. 440–455, 2020, doi: 10.1016/j.iref.2020.08.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1059056020301969

[39] S. Amershi *et al.*, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 291–300, doi: 10.1109/ICSE-SEIP.2019.00042. [Online]. Available: https://ieeexplore.ieee.org/document/8804457

[40] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 415–425, doi: 10.1109/ICSE.2015.60. [Online]. Available: https://ieeexplore.ieee.org/document/7194593

[41] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002, doi: 10.1613/jair.953. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10302

[42] C.-M. Teng, "A comparison of noise handling techniques." in *FLAIRS Conference*, 2001, pp. 269–273, university of West Florida, USA. [Online]. Available: https://www.aaai.org/Papers/FLAIRS/2001/FLAIRS01-052.pdf

[43] D. Dua and C. Graff, "UCI machine learning repository," archive.ics.uci.edu, http://archive.ics.uci.edu/ml, 2017, (accessed Jan. 17, 2023).

[44] G. Phillips, "Pairwise attribute noise detection algorithm for detecting noise in surface electromyography recordings," 2016, MSc Dissertation, Dept. of Electrical Computer Engineering, Univ. of New Brunswick, New Brunswick, Canada. [Online]. Available: https://unbscholar.lib.unb.ca/islandora/object/unbscholar%3A7747

[45] R. Merletti and P. Parker, "Surface emg applications in neurology," in *Electromyography: Physiology, Engineering, and Non-Invasive Applications*, 2004, ch. 12, pp. 323–342, doi: 10.1002/0471678384.ch12. [Online]. Available: https://doi.org/10.1002/0471678384.ch12

[46] ——, "Applications in exercise physiology," in *Electromyography: Physiology, Engineering, and Non-Invasive Applications*, 2004, ch. 14, pp. 365–379, doi: 10.1002/0471678384.ch14. [Online]. Available: https://doi.org/10.1002/0471678384.ch14

[47] J.-X. Zhong, N. Li, W. Kong, S. Liu, T. H. Li, and G. Li, "Graph convolutional label noise cleaner: Train a plug-and-play action classifier for anomaly detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1237–1246, doi: 10.1109/CVPR.2019.00133. [Online]. Available: https://ieeexplore.ieee.org/document/8953791

[48] Q. Wang, J. Zeng, and W. Song, "A new electromagnetism-like algorithm with chaos optimization," in *2010 International Conference on Computational Aspects of Social Networks*, 2010, pp. 535–538, doi: 10.1109/CASoN.2010.124. [Online]. Available: https://ieeexplore.ieee.org/document/5636954

[49] B. Kroll, D. Schaffranek, S. Schriegel, and O. Niggemann, "System modeling based on machine learning for anomaly detection and predictive maintenance in industrial plants," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–7, doi: 10.1109/ETFA.2014.7005202. [Online]. Available: https://ieeexplore.ieee.org/document/7005202

[50] K. R. Kini and M. Madakyaru, "Improved anomaly detection based on integrated multi-scale principal component analysis using wavelets: An application to high dimensional processes," *IFAC-PapersOnLine*, vol. 53, no. 1, pp. 398–403, 2020, 6th Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2020. doi: 10.1016/j.ifacol.2020.06.067. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320300860

[51] J. Risch and R. Krestel, "Bagging BERT models for robust aggression identification," in *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*. Marseille, France: European Language Resources Association (ELRA), May 2020, pp. 55–61. [Online]. Available: https://aclanthology.org/2020.trac-1.9

[52] G. H. Lee and S.-Y. Shin, "Federated learning on clinical benchmark data: performance assessment," *Journal of medical Internet research*, vol. 22, no. 10, p. e20891, 2020, doi: 10.2196/20891. [Online]. Available: https://www.jmir.org/2020/10/e20891

[53] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, pp. 1–13, Jan 2020, doi: 10.1186/s12864-019-6413-7. [Online]. Available: https://doi.org/10.1186/s12864-019-6413-7

[54] S. M. Saqlain *et al.*, "Fisher score and matthews correlation coefficient-based feature subset selection for heart disease diagnosis using support vector machines," *Knowledge and Information Systems*, vol. 58, no. 1, pp. 139–167, Jan 2019, doi: 10.1007/s10115-018-1185-y. [Online]. Available: https://doi.org/10.1007/s10115-018-1185-y

[55] D. T. Jones and J. J. Ward, "Prediction of disordered regions in proteins from position specific score matrices," *Proteins: Structure, Function, and Bioinformatics*, vol. 53, no. S6, pp. 573–578, 2003, doi: 10.1002/prot.10528.

[Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/prot.10528

[56] D. Borthakur, "HDFS," hadoop.apache.org, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, accessed Jan. 17, 2023).

[57] S. Vegas, "What makes a good empirical software engineering thesis?: Some advice," 2015, Accessed Jan. 13, 2023. [Online]. Available: https://ceur-ws.org/Vol-1469/intro1.pdf

[58] "Error log," techopedia.com, https://www.techopedia.com/definition/26306/error-log, (accessed Jan. 17, 2023).

[59] Anonymous, "Loghub," https://doi.org/10.5281/zenodo.3227177, Sep. 2021, Zenodo, (accessed Jan. 16, 2023).

[60] M. Blech, "Xmltodict," pypi.org, https://pypi.org/project/xmltodict/, (accessed Jan. 16, 2023).

[61] Y. Zhu, E. Zhong, Z. Lu, and Q. Yang, "Feature engineering for semantic place prediction," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 772–783, 2013, Mobile Data Challenge. doi: 10.1016/j.pmcj.2013.07.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119213000862

[62] S. Qaiser and R. Ali, "Text mining: use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018, doi: 10.5120/ijca2018917395. [Online]. Available: https://www.researchgate.net/publication/326425709_Text_Mining_Use_of_TF-IDF_to_Examine_the_Relevance_of_Words_to_Documents

[63] H. Schütze, C. D. Manning, and P. Raghavan, "Term frequency and weighting," in *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39, ch. 6.2, pp. 107–110, doi: 10.1017/CBO9780511809071. [Online]. Available: https://doi.org/10.1017/CBO9780511809071

[64] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 402–411, doi: 10.1109/ICSE.2013.6606586. [Online]. Available: https://ieeexplore.ieee.org/document/6606586

[65] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, doi: 10.48550/ARXIV.1201.0490. [Online]. Available: https://arxiv.org/abs/1201.0490

[66] F. Fylan, "Semi-structured interviewing," *A handbook of research methods for clinical and health psychology*, vol. 5, no. 2, pp. 65–78, 2005, NewYork: Oxford University Press.

[67] H. Kallio, A.-M. Pietilä, M. Johnson, and M. Kangasniemi, "Systematic

methodological review: developing a framework for a qualitative semi-structured interview guide," *Journal of advanced nursing*, vol. 72, no. 12, pp. 2954–2965, 2016, doi: 10.1111/jan.13031. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jan.13031

[68] L. Gren, R. Torkar, and R. Feldt, "Group development and group maturity when building agile teams: A qualitative and quantitative investigation at eight large companies," *Journal of Systems and Software*, vol. 124, pp. 104–119, 2017, doi: 10.1016/j.jss.2016.11.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216302266

[69] A. Steckler and K. R. McLeroy, "The importance of external validity," *American journal of public health*, vol. 98, no. 1, pp. 9–10, 2008, doi: 10.2105/AJPH.2007.126847. [Online]. Available: https://doi.org/10.2105/AJPH.2007.126847

[70] M. K. Slack and J. Draugalis, Jolaine R., "Establishing the internal and external validity of experimental studies," *American Journal of Health-System Pharmacy*, vol. 58, no. 22, pp. 2173–2181, 11 2001, doi: 10.1093/ajhp/58.22.2173. [Online]. Available: https://doi.org/10.1093/ajhp/58.22.2173