Automated error matching system using machine learning and data clustering

Evaluating unsupervised learning methods for categorizing error types, capturing bugs, and detecting outliers.

August Johnson and Jonatan Bjurenfalk

Supervisor : Chih-Yuan Lin
Examiner : Kristian Sandahl

External supervisor : Linus Lyckehult
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/her own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© August Johnson and Jonatan Bjurenfalk
Abstract

For large and complex software systems, it is a time-consuming process to manually inspect error logs produced from the test suites of such systems. Whether it is for identifying abnormal faults, or finding bugs; it is a process that limits development progress, and requires experience. An automated solution for such processes could potentially lead to efficient fault identification and bug reporting, while also enabling developers to spend more time on improving system functionality. Three unsupervised clustering algorithms are evaluated for the task, HDBSCAN, DBSCAN, and X-Means. In addition, HDBSCAN, DBSCAN and an LSTM-based autoencoder are evaluated for outlier detection. The dataset consists of error logs produced from a robotic test system. These logs are cleaned and pre-processed using stopword removal, stemming, term frequency-inverse document frequency (tf-idf) and singular value decomposition (SVD). Two domain experts are tasked with evaluating the results produced from clustering and outlier detection. Results indicate that X-Means outperform the other clustering algorithms when tasked with automatically categorizing error types, and capturing bugs. Furthermore, none of the outlier detection methods yielded sufficient results. However, it was found that X-Means’s clusters with a size of one data point yielded an accurate representation of outliers occurring in the error log dataset. Conclusively, the domain experts deemed X-means to be a helpful tool for categorizing error types, capturing bugs, and detecting outliers.
Acknowledgments

First of all, we would like to thank the Iota team at ABB Robotics for making us feel welcome during these six months. A special thanks to Per Muhr and Linus Lyckehult for contributing to the external evaluation, as well as providing valuable discussions.

We would also like to thank our supervisor at the University, Chih-Yuan Lin, and our examiner, Kristian Sandahl, for valuable input throughout the thesis.

Finally, we would like to thank our opponent Adrian Royo for providing valuable feedback during the opposition seminars.
# Contents

**Abstract** iii  
**Acknowledgments** iv  
**Contents** v  
**List of Figures** vii  
**List of Tables** viii  

1 **Introduction** 1  
   1.1 Aim 2  
   1.2 Research Questions 2  
   1.3 Delimitations 3  

2 **Theory** 4  
   2.1 Pre-processing 4  
   2.2 Dimensionality Reduction 5  
   2.3 Language Modeling 7  
   2.4 Data Clustering 8  
   2.5 Outlier Detection 13  
   2.6 Validation Methods 15  

3 **Related Work** 19  
   3.1 Failure analysis 19  
   3.2 System Log Analysis 20  
   3.3 Search Strategy 21  

4 **Method** 22  
   4.1 Overview 22  
   4.2 Environment 23  
   4.3 Dataset 25  
   4.4 Pre-processing 26  
   4.5 Implementation 29  
   4.6 Bug Matching 31  
   4.7 External Evaluation 31  

5 **Results** 33  
   5.1 Implementation 33  
   5.2 External evaluation 37  

6 **Discussion** 42  
   6.1 Results 42  
   6.2 Method 44
List of Figures

2.1 Illustration of data that is clustered into groups of similar objects. ........................................... 9
2.2 Illustration of DBSCAN .................................................................................................................. 10
2.3 K-means with K = 3. ....................................................................................................................... 12
2.4 X-means adding one more centroid to the dataset ........................................................................ 12
2.5 Illustration of a simple Feed-Forward Neural Network ................................................................. 13
2.6 Illustration of a simple Recurrent Neural Network ........................................................................ 14
2.7 Illustration of an LSTM-cell ......................................................................................................... 15

4.1 Illustration of Phase 1 ..................................................................................................................... 23
4.2 Illustration of Phase 2 ..................................................................................................................... 23
4.3 t-SNE visualization of the data ..................................................................................................... 28
4.4 Visualization of the data with Truncated SVD ............................................................................ 28
4.5 Illustration k-nearest neighbor graph .......................................................................................... 29

5.1 K-nearest neighbour graph ........................................................................................................... 33
5.2 Average reconstruction loss per epoch ......................................................................................... 36
5.3 Reconstruction loss per sample .................................................................................................... 36
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Illustration of stop word removal</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Illustration of stemming</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Illustration of noise removal</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Illustration of tokenization of a string using white space delimiter.</td>
<td>5</td>
</tr>
<tr>
<td>2.5</td>
<td>Illustration of the bag-of-words model</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>Structure of the error data file</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Structure of the combined dataframe</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Structure of the TF-idf matrix</td>
<td>27</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation of Explained variance ratio on TF-idf matrix</td>
<td>27</td>
</tr>
<tr>
<td>4.5</td>
<td>Structure of the result file</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Structure of the result file</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>Structure of the result file</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>S_dbw for DBSCAN configurations using euclidean distance</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>S_dbw for DBSCAN configurations using cosine distance</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>HDBSCAN configurations with euclidean distance</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>HDBSCAN configurations with an approximation of the cosine distance.</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>X-Means configurations</td>
<td>35</td>
</tr>
<tr>
<td>5.6</td>
<td>Algorithm rank: error type categorization</td>
<td>38</td>
</tr>
<tr>
<td>5.7</td>
<td>Algorithm rank: bug matching performance</td>
<td>40</td>
</tr>
<tr>
<td>5.8</td>
<td>Ranking: Outlier detection</td>
<td>41</td>
</tr>
</tbody>
</table>
Software maintenance is an integral part of a software development life cycle in the sense that it ensures that the solution stays consistent as technology and business environments change. It is a process of correcting previously undiscovered flaws, in conjunction with maintaining functionality and improving performance [1]. For large and complex software systems, identifying faults and reporting bugs in an efficient manner, have become increasingly difficult as more synthetic test cases and test runs are needed to be passed for verifying and validating. When test executions fail, they are usually analyzed by developers in order to identify root causes, and to find if an underlying bug has enabled such failures to occur. As complex systems can produce a large number of software errors, it is highly important for the failure analysis process to be as efficient as possible.

Previous studies have proposed several different approaches to make the failure analysis process more efficient. A study conducted by Q. Lin et al. [2] proposed a solution for clustering log data from a large-scale online service system. They utilized Agglomerative Hierarchical Clustering (AHC), pairing logs of high similarity to the same cluster. The solution reduced the required effort for the troubleshooting process of generated logs, allowing a more efficient root cause analysis. AHC considers each point to be its own cluster and then connects the clusters with its closest neighbour until only one cluster remains. Textual features can potentially be better modeled using density based clustering due to the fact that density based algorithms can capture arbitrarily shaped clusters. In a study especially interesting for this thesis, X. V. Nguyen et al. [3] proposed an automated and a semi-automated clustering approach for error messages produced from synthetic software test runs. They utilized a Support Vector Machine (SVM) classifier, for predicting error-categories from a labeled dataset consisting of 300 error messages. In addition, they utilized Density-based Spatial Clustering of applications with Noise (DBSCAN) for detecting outliers in the dataset, where found outliers were manually categorized. However, the study only took error messages as input data, where log files were not taken into consideration. According to W. Xu et al. [4] more sophisticated features can result in a more accurate analysis of error logs. Therefore, expanding the input data to include information from separate log files is of interest.

With new features and products being developed, new error types will arise. These errors will need to be handled separately since the clustering will only be able to categorize known
1.1 Aim

In order to capture the arbitrary shape of text data and make use of more information compared to previous studies, this thesis aims to provide an effective unsupervised learning model for accurately clustering error log data into different categories, using historical data produced from ABB Robotics’ internal testing systems. The goal is to achieve a solution that automates the process of linking error logs to known bugs, while also being able to identify if newly generated errors are potential outliers, or associated with a previously discovered cluster.

1.1. Aim

In order to capture the arbitrary shape of text data and make use of more information compared to previous studies, this thesis aims to provide an effective unsupervised learning model for accurately clustering error log data into different categories, using historical data produced from ABB Robotics’ internal testing systems. The goal is to achieve a solution that automates the process of linking error logs to known bugs, while also being able to identify if newly generated errors are potential outliers, or associated with a previously discovered cluster.

1.1 Aim

In this thesis, the aim will be to cluster error log data based on textual similarities, with the purpose of linking found clusters with known bugs. Several unsupervised clustering methods will be evaluated for the task. The evaluation process will be conducted by having the results validated by developers with expert-knowledge of the error log data. This is then combined with internal validation methods for each clustering algorithm, in order to ensure optimal algorithm performance. The density-based clustering algorithms Density-based Spatial Clustering of applications with Noise (DBSCAN) and Hierarchical DBSCAN (HDBSCAN) will be evaluated, as well as the partitional X-means algorithm. Such algorithms are especially suitable for instances where it is difficult to pre-define the number of clusters, which is the case for this particular problem, as the number of error-categories is not available.

In addition, as new features within a software system are developed and tested, new and previously unseen failures will appear as well. This creates the need for an automated solution which detects such instances, used in parallel with clustering known errors and linking them with bug reports. Such a solution will be implemented by evaluating several outlier detection methods, including the use of a recurrent neural network (RNN) based autoencoder in conjunction with DBSCAN’s and HDBSCAN’s outlier detection function.

1.2 Research Questions

In order to achieve the aim of this thesis, the following research questions will be answered.

1. How does HDBSCAN, DBSCAN, and X-means compare, when tasked with producing clusters that represent different error types?

HDBSCAN, DBSCAN and X-means are unsupervised learning algorithms that do not require a predefined number of clusters to be known. The choice of algorithms is motivated by the lack of domain-knowledge relating to defining the number of clusters prior to running the algorithms. This is due to the number of error types being unknown, in addition to the fact that new faults and bugs appear as new iterations of the robotic software and hardware is tested. In order to determine which of the algorithms yield the strongest results, they will be evaluated by domain experts with expert-knowledge of the systems and error types at ABB.

2. Is it possible to draw a connection between found clusters from the error log data with known bugs?

One of the primary aims of this thesis is to link errors with bug reports. Since the error log files contain extensive information regarding the sequence of events occurring prior to a failure, it would be interesting to investigate if failures with separate error messages share a root cause, and if this root cause is captured by the clustering algorithms. In order to determine this, found clusters containing faults caused by a bug can be evaluated.
1.3. Delimitations

against other faults contained within the same cluster. This evaluation process will be conducted by the domain experts at ABB.

3. Is an RNN-based autoencoder better suited for finding uncommon or abnormal error logs, when compared to the outliers found by DBSCAN and HDBSCAN?

As new updates are developed and tested, new and previously unseen errors are produced, which could potentially be caused by a bug. In order to detect such instances, an RNN-based autoencoder will be evaluated for outlier detection, together with DBSCAN and HDBSCAN’s functionality for outlier detection. Since RNNs are widely used within text mining, it would be interesting to evaluate how well such a model replicates error log data, and if datapoints that yield a high reconstruction error are outliers. In order to answer this research question, the results from each outlier-detection method will be evaluated by the domain experts at ABB, where they will investigate if the found outliers are actual uncommon errors or not.

1.3 Delimitations

The data used in this study is limited to the error log data found in the regression test software at ABB Robotics.
This chapter will present the theory required for conducting the study. The first section briefly introduce data pre-processing methods widely used in this research area, followed by a section on unsupervised clustering. The third section explains density-based clustering methods and describes in detail how the algorithms work. In addition, The outlier-detection methods are described, as well as the evaluation methods used for benchmarking algorithm performance.

2.1 Pre-processing

Pre-processing is the process of transforming data into a form that is predictable and analyzable when used as input for a given model. As noise in text can interfere with text analysis quality, it is important to remove such noise to ensure that the data is homogenic, resulting in a higher quality analysis. This section will cover the theory behind the different pre-processing methods used.

Text cleaning

Noise removal can be achieved in many different ways and depends greatly on the given domain and what is defined as noise for that specific domain. For instance, the underscore character might be highly important for some domain, while completely irrelevant for another. Common practices for text cleaning include removing special characters, digits, stopwords, text stemming, and transforming the text into lowercase. Stopwords are words which are insignificant in describing a text context. For example, in a search query "How to pre-process text-data?", it would be effective to retrieve "pre-process text data" and remove "how to", as it contains low information [5]. In english, commonly used stop words include "an", "the","is", and "are". Below is an illustration of how stop words are removed from a sentence.

<table>
<thead>
<tr>
<th>Table 2.1: Illustration of stop word removal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence</td>
</tr>
<tr>
<td>this is an example of stop word removal</td>
</tr>
</tbody>
</table>
Stemming is a commonly used technique within information retrieval and data mining. It works by removing suffixes from words, with the purpose of acquiring its stem. Stemming is especially good for contexts where word meaning isn’t of the highest priority. One drawback of stemming is that some of the produced morphological variants are not real words. Below is a table illustrating the stemming process.

<table>
<thead>
<tr>
<th>Original word</th>
<th>Stemmed word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>Process</td>
</tr>
<tr>
<td>Natural</td>
<td>Natur</td>
</tr>
<tr>
<td>Language</td>
<td>Languag</td>
</tr>
<tr>
<td>Lightweight</td>
<td>Lightweight</td>
</tr>
</tbody>
</table>

The table below illustrates how noise is completely removed from given sentences, by removing special characters, digits, applying stemming, and transforming the text into lowercase.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Cleaned Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 This_is_the_first!_Sentence</td>
<td>first sentence</td>
</tr>
<tr>
<td>1 Stemming processes transforms texts!</td>
<td>stem process transform text</td>
</tr>
<tr>
<td>2 Transformation_of_texts</td>
<td>transform text</td>
</tr>
</tbody>
</table>

**Tokenization**

Given a defined document, tokenization is the task of dividing the document into pieces of tokens. A token is a sequence of characters that are grouped together as a useful unit for processing, but is not necessarily a word. A type signifies all unique tokens in the document, representing a class of the same character sequence. The types in a document defines the vocabulary, which contains all unique tokens. [5]

<table>
<thead>
<tr>
<th>Input: a sofa is a furniture</th>
<th>Tokens: a sofa is a furniture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types: a sofa is furniture</td>
<td></td>
</tr>
</tbody>
</table>

**2.2 Dimensionality Reduction**

A major step in any scientific study is to identify patterns from collected data in order to support a general claim. For two and three-dimensional data, such patterns can be identified through simply visualizing the data. However, in many practical instances, the data-set will contain more than three variables, making it difficult for a human to understand through just visualization alone. In order to resolve this, there exists mapping algorithms which reduces data dimensionality, while preserving relationships between the data-points. Although some information is lost when reducing data dimensionality, these mapping algorithms aim at preserving the most useful information regarding the data. In this segment, the theory behind t-distributed Stochastic Neighbor Embeddings (t-SNE) and Singular Value Decomposition (SVD) is presented. The segment begins with an introduction to Stochastic Neighbor Embeddings (SNE), which t-SNE is based on.
SNE
First introduced by G. Hinton and S. Roweis [6], SNE is a probabilistic approach for embedding high-dimensional objects into low-dimensional space. It is a Gaussian model, centered on every data point in the high-dimensional space. The densities define a probability distribution over all potential neighbors of a given object, which follows a normal distribution. The algorithm computes the asymmetric probability \( p_{ij} \) for each object \( i \) that would select \( j \) as a neighbor [6]

\[
p_{ij} = \frac{\exp(-d_{ij})}{\sum_{k \neq i} \exp(-d_{ik})}
\]

where \( d_{ij} \) is the dissimilarity, which could be given by the problem definition, or computed using the squared euclidean distance [6]

\[
d_{ij}^2 = \frac{||x_i - x_j||^2}{2\sigma_i^2}
\]

where \( \sigma_i \) is computed using binary search, for finding the \( \sigma_i \) where the Shannon entropy of the neighbors’ distribution is \( \log k \). \( k \) is the perplexity, which is a measure for the number of local neighbors, and is a tunable parameter [6]. The Shannon entropy measures the expected amount of information in an event, drawn from a distribution [7].

In low-dimensional space, the probability \( q_{ij} \) for point \( i \) selecting \( j \) as its neighbor is computed in a similar manner as in the high-dimensional space [6]

\[
q_{ij} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq i} \exp(-||y_i - y_k||^2)}
\]

The Kullback-Leibler divergences cost function is used for preserving the distances to nearby points, making \( p_{ij} \) and \( q_{ij} \) match in the best possible way. [6]

\[
C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}
\]

The cost function can be minimized in several ways, including the gradient descent method used for finding local optima.

One of the main issues with SNE is the crowding problem. It is a result of higher dimensions having more space for neighboring points, compared to smaller dimensions (2D or 3D). This results in the neighboring data points being clumped together in order to fit the smaller dimensional space. Another drawback of SNE is that the asymmetric cost function is difficult to optimize, which is solved by the introduction of t-SNE.

t-SNE
t-SNE is a non-linear dimensionality reduction algorithm developed by L. Maaten and G. Hinton [8], and is a modified version of SNE. As SNE, it is suitable for visualizing high-dimensional data in a two or three-dimensional space. It takes high-dimensional distances between data points with a given metric, and converts them into probabilities that represent low-dimensional similarities [8]. One of the main differences is that the cost function for t-SNE is symmetric, enabling simpler gradient descents and counteracts the complexity problem of SNE’s cost function. In addition, the student t-distribution is used for the low-dimensional space, as its heavier tail mitigates the crowding problem enabled by the normal distribution [8].

The Kullback-Leibler cost function is also used for t-SNE, but with \( p_{ii} \) and \( q_{ii} \) set to zero. For every \( i \), \( p_{ii} = p_{ji} \) and \( q_{ii} = q_{jj} \), making t-SNE symmetric. [8]
Singular Value Decomposition

Singular Value Decomposition (SVD) is a matrix factorization technique that can be used to reduce the dimensionality of a dataset. The algorithm factors an $M \times N$ matrix $C$ into three component matrices $U$, $S$ and $V^T$, see equation (2.5). Given $C$ with rank $r$, $U$ is an $M \times M$ matrix, where $S$ is an $M \times N$ matrix and $V$ is an $N \times N$ matrix. $V^T$ is the transpose of $V$. $S$ contains the square root of the eigenvalues of $C^T \times C$ in its diagonal, referred to as the singular values of $C$. The singular values of $C$ can be used to understand the variance in the original data explained by each vector. Given the decomposition of $C$, the original matrix can be reconstructed. [5]

$$C = U \times S \times V^T$$ (2.5)

In written form, it is conventional to omit the columns containing zeros in the decomposed matrices of $C$. $U$ is reduced to a $M \times r$ matrix, $S$ to an $r \times r$ matrix and $V$ to an $N \times r$ matrix. This written form is referred to as truncated or reduced SVD. [5]

Latent Semantic Analysis

Latent Semantic Analysis (LSA) was introduced as a technique for improving information retrieval by reducing dimensionality [9]. Besides the use in information retrieval, it has also been applied in document clustering. H. Schütze and C. Silverstein showed that LSA significantly improved distance calculation speed in clustering, while not affecting the quality of the clusters adversely [10].

The LSA method consists of four main steps, where the first two are similar to the ones used in vector space modeling. [9]

1. **Term-Document Matrix**: A collection of documents or text strings are represented as a matrix, with unique words as columns and documents as rows.

2. **Transformed Term-Document Matrix**: instead of using the term frequency, the matrix is transformed. Typically with inverse document frequency or an entropy based score.

3. **Dimension Reduction**: SVD is performed on the transformed term-document matrix, where the $k$ singular values with the largest score is retained. Each term and document will be represented as a $k$-dimensional vector, where the SVD representation is a $k$-dimensional approximation of the original space.

4. **Retrieval in Reduced Space**: Similarities are computed in the reduced space, rather than in the original space.

2.3 Language Modeling

The bag-of-words model is a model for simplifying representations of texts, where the sentence or document is represented as the bag of its words. This model keeps the number of word occurrences, but disregards the order of words and grammar. The model captures the idea that not all words are equally important in a text, with the intuition that two documents with similar bag-of-word representations are similar in content. Two documents containing the same word are more similar than two documents that do not, and the more words two documents have in common, the more similar they are. [5]

The representation of documents as vectors that captures the relative importance of its terms is known as a Vector Space Model. Each term $t$ in a document $d$ is assigned a weight $w_t$, which depends on the term’s frequency in the document. A score can then be computed for term $t$
in document \( d \), based on the weight \( w \). The simplest approach is to set \( w \) equal to the number of occurrences of \( t \) in \( d \), called term frequency (tf). [5]

Table 2.5 illustrates two similar sentences with different meanings, which are treated as equal, using a simple bag-of-words approach.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Bag of Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 the man bites the dog</td>
<td>the:2, man:1, bites:1, dog:1</td>
</tr>
<tr>
<td>1 the dog bites the man</td>
<td>the:2, man:1, bites:1, dog:1</td>
</tr>
</tbody>
</table>

**Term frequency - Inverse Document Frequency**

Term frequency - Inverse Document Frequency (TF-IDF) is a statistical measure for evaluating word relevance in a document contained in a collection of documents [11]. Term frequency (TF) is proportional to the number of times a word occurs in a document, and is offset by the number of documents in which the word is present. This offset enables words that frequently appear in all documents to have a low TF-IDF score. TF-IDF is achieved by multiplying the frequency of a given word \( t \), with the inverse document frequency (idf) for the word across a collection of \( n \) documents. [12]

\[
\text{TF-IDF} = \text{tf}(t, d) \cdot \text{idf}(t) \tag{2.6}
\]

where,

\[
\text{idf}(t) = \log\left(\frac{n}{df(t)}\right) + 1 \tag{2.7}
\]

TF-IDF is commonly used within the field of information retrieval and text mining, where it can be utilized as a feature engineering tool when summarizing texts and classifying documents.

### 2.4 Data Clustering

Data clustering is a method for grouping a set of data points into clusters, where data points within one cluster share a high similarity, but are dissimilar to points contained in other clusters. Clustering is an unsupervised machine learning method for finding underlying structures in a given data-set, without the need of having labels to be associated with the data points. For clustering problems, algorithms can either perform hard or fuzzy clustering. Hard clustering algorithms assign one cluster class to each object, with the assumption that there only exists one cluster for every data point. In Fuzzy clustering, data points can belong to one or several clusters, which is done by relaxing the hard clustering assumption and assigning data points to clusters using probabilities. [13]

In order to separate the different clusters, similarities and distances are used. Without this measure, the task of performing a cluster analysis becomes meaningless since there is no way to distinguish the clusters. Similarity metrics are used to describe how similar two objects are, where a higher value signifies a higher similarity. The opposite is true for distance, as a greater value corresponds to less similar data points. [13]

A density-based cluster is a set of objects in the data space, spread over a contiguous region. High density regions are considered to be clusters, separated by low density regions. Objects in low density regions are considered to be outliers or noise. Density-based clustering does
not make assumptions about the variance within a cluster or the density. Therefore, density-based clusters can be arbitrarily shaped. Neither does the density-based methods require the number of clusters as input, making the method non parametric. [14]

Figure 2.1: Illustration of data that is clustered into groups of similar objects.

**Distance Measures**

In document clustering, applying different distance measures can result in different clustering results. Therefore, the distance measure is an important choice, with which the outcome can be influenced. [5]

**Euclidean Distance**

One common distance measure often used in document clustering is the euclidean distance, where the distance between two vector representations $\vec{X}$ and $\vec{Y}$ is computed. [5]

$$|\vec{X} - \vec{Y}| = \sqrt{\sum_{i=1}^{M} (x_i - y_i)^2}$$ (2.8)

**Cosine Similarity**

Two documents $d_1$ and $d_2$ with similar content can have significant vector differences if they vary in length. The standard way of compensating for the effect of document length when computing the similarity between two documents is to compute the cosine similarity of $\vec{V}(d_1)$ and $\vec{V}(d_2)$. [5]

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{||\vec{V}(d_1)|| ||\vec{V}(d_2)||}$$ (2.9)

**DBSCAN**

Density Based Spatial Clustering of Applications with Noise (DBSCAN), first introduced by M. Ester et al. [15], is a clustering algorithm that discovers clusters based on data-point density. It was proposed as a solution for instances where the required domain knowledge for defining the number of clusters isn’t present. In order to construct a dense region, DBSCAN requires epsilon distance $\epsilon$ and the minimum number of points ($\text{minPts}$) to be defined. There exists two kinds of points inside a cluster, a core point at the center, and border points on the border of the cluster. A point is set as a core point $p_c$ if $\text{minPts}$ amount of points are within $\epsilon$ distance to $p_c$. If a point $p$ is reachable from some other point through a series of core points, they are density-reachable. If two points $p_1$, $p_2$ share a density reachable core point $p_c$, they are density-connected. A point can also be direct density-reachable if it has a core point in its neighbourhood. A cluster is formed when $p_c$ is density-reachable from $\text{minPts}$ number of points $p$, where $p$ is density-connected to $p_c$. [15]
2.4. Data Clustering

The figure above illustrates the clustering process of DBSCAN. MinPTS = 3, where Border point 1 is density-connected to Border point 2 through the core point.

HDBSCAN

Hierarchical Density Based Spatial Clustering of Applications with Noise (HDBSCAN), proposed by R. J. Campello et al., is an extension of DBSCAN where the requirement of determining the \( \epsilon \) parameter is removed. Hence, the only input parameter is \( m_{pts} \) which can be viewed as the minimum amount of data points needed to form a cluster. The algorithm aims to combine the benefits from both hierarchical clustering and density-based clustering by converting DBSCAN into a hierarchical clustering algorithm, and then extract a simplified tree containing the main clusters. [16]

The algorithm defines the core distance \( d_{core}(x_p) \) of an object \( x_p \in X \) as the distance from \( x_p \) to its \( m_{pts} \)-nearest neighbor. The Mutual Reachability Distance between two objects \( x_p \) and \( x_q \) is defined as [16]

\[
d_{mreach}(x_p,x_q) = \max(d_{core}(x_p),d_{core}(x_q),d(x_p,x_q))
\]  

(2.10)

where \( d(x_p,x_q) \) is the distance between the two objects. The Mutual Reachability Graph \( G_{m_{pts}} \) is a complete graph, in which the Mutual Reachability Distance between the respective pairs of objects represent the weight of each edge, where the objects in \( X \) are vertices. With the definitions for core distance, Mutual Reachability Distance and Mutual Reachability Graph, the main steps of the HDBSCAN algorithm can be described. [16]

1. Using the parameter \( m_{pts} \), compute the core distance for all data points in the dataset.
2. Compute a Minimum Spanning Tree (MST) of the Mutual Reachability Graph.
3. Extend the MST to $MST_{ext}$, by adding a "self edge" for each vertex with the core distance of the corresponding data point as weight.

4. Extract the HDBSCAN hierarchy as a dendrogram from the $MST_{ext}$:

   1. For the root of the $MST_{ext}$, assign all data points the same label.
   2. In decreasing order of the weights, iteratively remove all edges. If more than one edge have the same weight, the edges have to be removed in the same iteration:
      1. Before removing an edge, set the scale value of the current hierarchical level to the weight of the edge being removed.
      2. After removing an edge, assign labels to the connected component(s). If a component has at least one edge, assign it a new cluster label. Otherwise, assign it as noise.

K-means

The K-means algorithm is a partitional clustering method designed to cluster numerical data, where each cluster has a mean centroid. For a given number of initial clusters $K$, the data points are allocated to the nearest cluster. Thereafter, the cluster memberships are iteratively changed according to the error function, as described in equation (2.11). The algorithm stops executing when the error function does not improve, or when the membership of the clusters do not change. K-means assumes the number of clusters $K$ to be fixed and can be described as follows [13]

While there are no changes in cluster memberships:

1. Compute distance between cluster centroid and point.
2. Assign point to cluster given the arg(min) error.
3. Recompute cluster means of any changed cluster.

\[
Error = \sum_{i=1}^{K} \sum_{x \in C_i} d(x, \mu(C_i)) \tag{2.11}
\]

Where $\mu(C_i)$ is the centroid of cluster $C_i$, where $d$ can be any distance function. $d(x, \mu(C_i))$ is the distance between $x$ and the centroid.

The algorithm has some important properties. Since the complexity is linearly proportional to the size of the data-set, it is efficient in clustering large data-sets. Furthermore, the algorithm stops execution at a local optimum and the clusters have a convex shape. The performance depends on the starting centroids. [13]

The starting centroids can be selected at random. However, since the algorithm is fully deterministic given the starting centroids, the performance will be bad if the centers are poorly selected. [13]

X-means

X-means is an algorithm for efficiently finding the best $K$ for a given data-set, when using k-means. Instead of having to specify $K$, the user will specify a reasonable range for $K$. The algorithm starts with $K$ equal to the lower bound for $K$. It will add centroids where it is needed, until the upper bound is reached. [17]
2.4. Data Clustering

The algorithm consists of two actions performed in a loop until the upper bound is reached. The first action *Improve-Params*, runs the K-means algorithm until convergence. The second action *Improve-Structure*, finds out if and where new centroids should appear. [17]

The *structure improvement* action will split each centroid into two children, which are moved with a distance proportional to the size of the region in opposite directions along a random vector. Next, a local K-means with \( K = 2 \) is performed for each child. Model selection is then performed on all children pairs, with the purpose of finding out if the children improves model performance, or if the parent captures the distribution equally well. Centroids that already have a cluster in the underlying distribution will not be modified. Hence, regions of space not represented sufficiently by the centroids will be given more attention, as the number of centroids in them will increase. With this method, all splitting configurations will be covered in the search space, deciding which configuration to explore by improving the BIC score in each region. [17]

![Figure 2.3: K-means with \( K = 3 \).](image)

![Figure 2.4: X-means adding one more centroid to the dataset.](image)

Every K-means model that X-means evaluates, contain different values of \( K \). The best model is found by evaluating the Bayesian Information Criterion (BIC) for every model [17].

**Bayesian Information Criterion**

The Bayesian Information Criterion (BIC) is a criterion for model selection for a finite model-set proposed by G. Schwarz [18]. Given some data \( D \) and a set of alternative models \( M_j \), posterior probabilities are used for scoring each model. Spherical Gaussians are assumed for K-means, where X-means use the following formula for approximating the posteriors [17]

\[
BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2} \log R
\]

(2.12)

Where \( \hat{l}(D) \) is the maximum log-likelihood for every model \( j \), and \( p_j \) is the number of parameters in the model, where \( R = |D| \). Limiting the log-likelihood to a set of points \( D_n \), \( 1 \leq n \leq K \), belonging to centroid \( n \).

\[
\hat{l}(D_n) = -\frac{R_n}{2} \log(2\pi) - \frac{R_n \cdot M}{2} \log(\hat{\sigma}^2) - \frac{R_n - K}{2} + R_n \log R_n - R_n \log R
\]

(2.13)

Under the identical spherical Gaussian assumption, the maximum likelihood estimate for the variance is

\[
\hat{\sigma}^2 = \frac{1}{R - K} \sum_i (x_i - \mu_i)^2
\]

(2.14)

The BIC score is used globally once X-means selects the best model, and locally in all centroid split tests. To extend the formula to consider all centroids instead of one, the log-likelihood for all individual centroids are summed, and \( R \) is replaced with the total number of points.
2.5 Outlier Detection

Outlier detection or Anomaly detection, is the process of detecting instances in the data which greatly deviate from the majority of instances present in the data.

Deep Learning for Outlier Detection

In recent years, deep learning has been proven to have strong capabilities in learning expressive representations of complex data. This can be achieved by using autoencoders, where a neural network is trained at replicating the input data, by minimizing the replication error. This results in commonly occurring instances to be easier for the network to replicate compared to less common instances. In such cases, the instances that yield a high reconstruction error, beyond a defined threshold, can be identified as outliers.

Artificial Neural Networks (ANN)

Artificial Neural Networks is one of the most widely used neural network designs, created with the initial purpose of simulating the way a biological neural network analyzes and processes information. It has become a foundation for the majority of existing neural network architectures. Examples of such architectures include feed-forward neural networks and recurrent neural networks.

Feed-forward Neural Network

A Feed-forward neural network consists of neuron-inspired processing units, often referred to as nodes. The nodes are organized in layers, where every node is connected to all nodes present in the previous layer. The formed connection between the nodes consists of a weight, which encodes the knowledge of the network. The input data will enter the first layer, known as the input layer, and thereafter pass through each of the other layers until it arrives at the final layer, known as the output layer. The layers between the input layer and output layer, are known as hidden layers [19]. The name feed-forward stems from the connections only moving in one direction. The figure below illustrates a simple feed-forward neural network for learning $y_1, y_2$ with the given inputs $X_1, X_2, X_3$.

![Illustration of a simple Feed-Forward Neural Network.](image-url)
2.5. Outlier Detection

Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of ANN that during training, takes sequences of inputs. An RNN contains an internal memory, enabling the network to learn from previous experiences, as well as to be precise in predicting what’s coming next [19]. In RNNs, the information is cycled through a loop, enabling decisions to be made based on current inputs and what it has learned from the inputs. Weights are applied to both current and previous inputs, where they are adjusted over time through gradient descent and back-propagation [19]. There exists two major issues regarding RNNs, the first one being when it assigns the weights with extremely high importance, which is known as the exploding gradient problem. This results in an unstable model, with an inability to learn. The second issue is known as the vanishing gradient problem, in which the gradient values become very small, preventing the weights from changing and hindering the model from learning more. The figure below illustrates a recurrent neural network for learning $y_1, y_2$ with the given inputs $X_1, X_2, X_3$.

![Illustration of a simple Recurrent Neural Network](image)

Figure 2.6: Illustration of a simple Recurrent Neural Network.

Long Short-Term Memory

The Long Short-Term Memory (LSTM) architecture was first introduced by S. Hochreiter and J. Schmidhuber [20], and is an extension of conventional RNNs. It handles the exploding and vanishing gradient problem by having self-loops within each LSTM-cell, which form paths that counteract the issue. Compared to conventional RNNs, an LSTM network consists of LSTM-cells, which are able to control how its long-term memory should store, forget, and output information. The long-term memory is known as the cell state $C$, which holds information throughout the training process. $C$ is updated during training, by having gates determine an input’s importance based on previous experiences.

An LSTM-cell consists of a forget gate, input gate, and an output gate. The forget gate will determine which information to delete from the cell state. This is done by taking the previous output $h_{t-1}$ and the current input $X_t$, where a sigmoid function will output a value between 0 and 1 for every value in the previous cell state $C_{t-1}$. Values that are closer to 0 will be forgotten, where values closer to 1 will be kept. In order to determine which information to take into account, the input gate is used. It will pass the previous output $h_{t-1}$ into a sigmoid function, which will yield values between 0 and 1 representing how important a
2.6 Validation Methods

given value is. \( h_{t-1} \) and the current input \( X_t \) is thereafter passed into a hyperbolic tangent function \( \tanh \), which will transform the values to be between -1 and 1. The outputs from the sigmoid function and \( \tanh \) are then multiplied for determining which information to keep. Finally, the output gate will determine what the next output \( h_t \) should contain. This is done by passing \( h_{t-1} \) and \( X_t \) into a sigmoid function. The cell state is thereafter passed into a \( \tanh \) function, where the resulting output is multiplied with the output from the sigmoid function. The new cell-state \( C_t \) and output \( h_t \) is thereafter passed into the next iteration. \[20\] Figure 2.7 illustrates an LSTM-cell.

![Figure 2.7: Illustration of an LSTM-cell](image)

Autoencoders

An Autoencoder is an artificial neural network used for unsupervised learning applications. Autoencoders have mainly been used within the area of dimensionality reduction and feature learning, although recent research has brought it to the forefront of generative modelling.

The purpose of an autoencoder is to learn how to reconstruct the input data. It consists of an encoder function \( h = f(x) \) and a decoder \( r = g(h) \). The encoder will encode the input to a hidden layer \( h \), constructing a latent representation of the data. Thereafter, the decoder will reconstruct the latent representation to an output. Autoencoders are usually designed to produce an approximate copy of the data, restricting them to only prioritize certain aspects of the input. This enables them to only take the most useful properties into consideration. \[19\]

2.6 Validation Methods

When analyzing and validating clustering results, several aspects has to be taken into account for improving algorithm performance. It involves determining the optimal number of clusters, and evaluating cluster quality without external information, as well as determining the tendency of the data. In addition, it also involves comparing clustering results with external information, and evaluating the results against the output from other clustering algorithms \[21\]. Cluster validation methods can be divided into three main categories: external, internal and relative. The external and internal approaches involve statistical testing, while the relative approach involve non-statistical testing. Since clustering is an unsupervised process, there are no labels or examples that can show the validity of the clusters found by a specific algorithm \[13\]. Rendering the external methods non-applicable for data-sets lacking labels or external data \[21\].
The internal metrics evaluate the clusters resulting from an algorithm using only features from the data set [13]. Partitional algorithms use a criteria based on cohesion and separation, while hierarchical algorithms normally use the cophenetic coefficient [21]. The popular silhouette coefficient is an example of a criteria for partitional algorithms [22]. The relative metrics pre-defines a criteria and uses a set of parameters for a specific algorithm, in order to decide which clustering result scores the best. The set of parameters that produces the best clustering result is selected [13].

\textbf{S\_Dbw}

S\_Dbw was proposed by M. Halkidi and M. Vazirgiannis [23] as a criteria for relative validation of clustering algorithms. The index enables optimal hyperparameter selection by considering the clusters’ compactness and the density between clusters. [23]

Inter-cluster density evaluates the average density between regions in relation to the density of the regions. It is computed as follows [23]

\[
\text{Dens\_bw}(C) = \frac{1}{C(C - 1)} \sum_{i=1}^{C} \sum_{j=1, j\neq i}^{C} \frac{\text{density}(u_{ij})}{\max(\text{density}(v_i), \text{density}(v_j))}
\] (2.15)

Where \(v_i\) and \(v_j\) are the centers of clusters \(C_i\) and \(C_j\), and \(u_{ij}\) is the point between the two clusters. The density is defined by as follows [23]

\[
\text{density}(u) = \sum_{i=1}^{n_{ij}} f(x_i, u)
\] (2.16)

Where \(n_{ij}\) is the number of tuples that belong to clusters \(C_i\) and \(C_j\). The function \(f(x_i, u)\) in \(2.16\) represents the number of points in the vicinity of a data point \(u\). It is defined as 0 if the distance between \(x\) and \(u\) is larger than the standard deviation of the clusters, and 1 otherwise.

Intra-cluster variance describes the average scattering of clusters.

\[
\text{Scat}(C) = \frac{1}{C} \sum_{i=1}^{C} \frac{\|\sigma(V_i)\|}{\|\sigma(S)\|}
\] (2.17)

Where \(\sigma(S)\) is the variance of the data set and \(\sigma(S)\) the variance of cluster \(c_i\).

By using the inter-cluster density and the intra-cluster variance, the index can be calculated. A lower score is equal to a better clustering result.

\[
S\_Dbw(C) = \text{Scat}(C) + \text{Dens\_bw}(C)
\] (2.18)

A drawback with S\_Dbw is that it does not work properly with arbitrarily shaped clusters, for example non-convex clusters [23]. Tong et al. [24] proposed an improvement of the S\_Dbw index, \(S\_Dbw_{new}\) which can handle non-circular clusters. The new criteria outperforms its predecessor, especially when the real data is non-circular or sparse. To overcome this problem, the new index introduces a method for finding a more precise point which represent the region between two clusters, instead of using the middle point.

\textbf{Density-Based Clustering Validation}

Several relative validity criteria have been proposed for globular clusters, for example \(C\_Dbw\) [25] and \(S\_Dbw\). \(S\_Dbw\), as many other relative validity criteria, consider the center of a cluster for computations. In arbitrarily shaped clusters the center might not be a representative point. \(C\_Dbw\) considers multiple representative points per cluster instead of one,
2.6. Validation Methods

the center. This enables it to handle arbitrarily shaped clusters. However, the criteria requires a fixed set of points, which is a drawback, since clusters can be of different sizes and shapes [26].

D. Moulavi et al. introduced Density-Based Clustering Validation (DBCV) to address the drawbacks of previous criteria. DBCV computes the most dense regions between clusters and the least dense regions within clusters, using Hartigan’s model of density contour trees [27]. By using the two measures, the connectedness of clusters can be computed. [26]

In order to calculate the core distance of an object, the all-points-core-distance $a_{pts\text{-coredist}}$ is computed as follows [26]

\[
\begin{align*}
a_{pts\text{-coredist}}(o) = \left( \frac{\sum_{i=1}^{n}(\frac{1}{KNN(o,i)})^d}{n_i - 1} \right)^{-\frac{1}{d}}
\end{align*}
\]

$a_{pts\text{-coredist}}$ is the inverse density of an object $o$ in a cluster $C_i$. The $a_{pts\text{-coredist}}$ variable is then applied for computing the Mutual Reachability Distance (MRD) for all objects in $C_i$. MRD is used to build a Minimum Spanning Tree $MST_{MRD}$ for the cluster $C_i$. This process repeats for all clusters, which results in one $MST_{MRD}$ for each cluster. By using the $MST_{MRD}$ with two definitions of density sparseness and density separation, the DBCV index can be calculated. The following definitions describe the density sparseness of a cluster (DSC) and the density separation of a cluster (DSPC) [26]

- Density sparseness of a cluster (DSC) is defined as the maximum edge of its $MST_{MRD}$.
- Density separation of a cluster (DSPC) with respect to another cluster is defined as the minimum MRD between the objects in the two clusters.

DSC can be interpreted as the lowest density area within a cluster, where DSPC can be interpreted as the area with the maximum density between two clusters. Using DSC and DSPC, the density-based quality of a single cluster ($V(C_i)$) can be computed. If a cluster has better DSC than DSPC, the validity index will be a positive value, and if the density in a cluster is lower than the density between two clusters, the index will be a negative value [26]

\[
V_C(C_i) = \frac{\min_{1 \leq j \leq l, j \neq 1} (DSC(C_i, C_j)) - DSC(C_i)}{\max(\min_{1 \leq j \leq l, j \neq 1} (DSPC(C_i, C_j)), DSC(C_i))} 
\]

(2.20)

With the density-based quality of a single cluster ($V(C_i)$), the size of a single cluster $|C_i|$ and the total number of objects, including noise, $|O|$ DBCV can be computed [26]

\[
DBCV(C) = \sum_{i=1}^{l} \frac{|C_i|}{|O|} V_C(C_i)
\]

(2.21)

Silhouette Coefficient

The silhouette coefficient was proposed by P. Rousseeuw for interpretation and validation of cluster analyses using partitioning techniques, like the K-means algorithm. The coefficient uses the comparison of cohesion and separation in the data to show which points lie within their cluster, and which points are between clusters. [22]

The silhouette coefficient can be computed using the following steps [21]
2.6. Validation Methods

1. For each point in the data set, compute the average distance \( a(i) \) to all other points in the same cluster

\[
a(i) = \frac{1}{|C_a|} \sum_{j \in C_a, j \neq i} d(i, j)
\]  

(2.22)

2. For each point in the data set, compute the minimum average distance \( b(i) \) between the point and all points not within the same cluster

\[
b(i) = \min_{C_b \neq C_a} \frac{1}{|C_b|} \sum_{j \in C_b} d(i, j)
\]  

(2.23)

3. For each point, compute the silhouette coefficient

\[
s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}
\]  

(2.24)

For each point in the data-set, the silhouette coefficient is defined in the interval \([-1, 1]\). The global coefficient is the summation of all individual points

\[
S = \frac{1}{n} \sum_{i=1}^{n} s(i)
\]  

(2.25)

The silhouette coefficient \( S \) indicate high separation between clusters for positive values and overlapping clusters for negative values. If the coefficient is zero the data is uniformly distributed throughout the euclidean space. [22]
3 Related Work

This chapter will present papers that are related to this thesis. The first section covers the area of failure analysis, in which it presents previous works related to automatic categorization of errors. The second section covers previous research related to system log analysis, in which machine learning has been applied for outlier detection and clustering.

3.1 Failure analysis

Failure analysis is an active research area, where both supervised and unsupervised learning methods are being explored. Previous studies fetch information from internal sources with the purpose of identifying error types and underlying data structures, either through means of classification or clustering. Examples of internal sources include execution profile features, source code and error messages.

W. Dickinson et al. [28] experimentally evaluate the feasibility of using cluster analysis for detecting failure causes from test case execution profiles. Their study compares different filtering procedures for selecting executions, where each filtering procedure has a sampling strategy and a similarity measure. The clustering algorithm used for all procedures is agglomerative clustering, because of its speed compared to partitional methods. The study concludes that filtering procedures based on clustering results are more effective than random sampling, and that similarity measures that give extra weight to unusual profile features are more effective. One of the main differentiating areas between this paper and ours, lie in them using agglomerative clustering for detecting failures, while this thesis aims to categorize error logs and detecting unusual instances of them. What makes this paper interesting is that this research was conducted in 2001. It gave insights into how failure analysis through means of clustering was performed in the early years of data science, and how it was achieved without the modern libraries and algorithms used today.

A. Podgurski et al. [29] expands on the analysis of execution profiles by proposing automated support for classifying reported software failures, involving the use of supervised and unsupervised pattern classification. Execution profiles typically have thousands of features. Therefore, logistic regression is used to select a subset of features, as it removes features that are linearly dependent or non-informative. The study discusses the difficulty of pre-
3.2 System Log Analysis

determining the number of clusters and suggests multivariate visualization in conjunction with clustering, in order to visually inspect the resulting clusters. Results show that the clustering algorithm creates a few large clusters containing sub-clusters with the same cause, indicating that the strategy can be effective in grouping failures with similar causes. One key difference between this papers and ours, is the use of supervised pattern classification prior to clustering and visualization. In order to reduce dimensionality, this thesis utilizes unsupervised dimensionality reduction opposed to supervised learning. Furthermore, the paper evaluates k-medoids as a clustering technique. This thesis evaluates a combination of density based and hierarchical clustering methods, in conjunction with x-means.

N. DiGiuseppe and J. A. Jones [30] propose a failure clustering technique based on execution semantics opposed to execution profile features. They hypothesise that semantically rich execution information can improve clustering results when automatically categorizing failures with the purpose of isolating causes. The study examines Latent-Semantic-Analysis (LSA) to categorize the semantic concepts of the executed source code by applying hierarchical agglomerative clustering. The study concludes that clustering based on semantic concepts is better and more precise than clustering execution profile features. The paper applies TF-IDF for pre-processing. However, the paper does not reduce dimensionality before applying the clustering algorithm. Therefore it is unclear if LSA is used, since the method requires the reduction of a term frequency matrix.

V. X. Nguyen et al. [3] propose an automated and a semi-automated error clustering method, with the purpose of grouping root causes for facilitating debugging and maintenance. Error messages are vectorized using TF-IDF and outliers are removed using DBSCAN, before evaluating Naive-Bayes and Support Vector Machines (SVM) for classification. Results indicate that the technique of clustering root causes using error messages is effective. Similarly to this thesis, the paper pre-processes error messages and uses TF-IDF for DBSCAN. However, the paper only applies the algorithm for outlier detection, and instead classifies new errors to known clusters with supervised learning. Furthermore, this thesis expands the dataset by using information from separate log files.

3.2 System Log Analysis

This segment covers related work within the area of unsupervised system log analysis. W. Xu et al. [4] propose a general methodology for mining information in system logs in order to automatically detect system runtime problems. The study parses console logs by combining source code analysis with information retrieval methods, where the unstructured log data is converted into structured features. Each feature corresponds to a different message type and has the term frequency as value. Principal Component Analysis (PCA) is applied to the features for anomaly detection and visualization, with the purpose of classifying each feature vector as normal or abnormal. Results indicate that the more sophisticated features constructed using console logs produce an accurate analysis, able to detect anomalies. Similarly to this thesis, the paper uses a bag of word representation of information from internal systems. However, instead of combining and pre-processing message strings, message types are identified and used as features. Since the message type is used as features there is no need to remove stop words or use TF-IDF, which is used in our work. Furthermore, the frequency matrix in the paper is not used for clustering, but rather for identifying outliers in need of investigation.

Q. Lin et al. [2] propose the framework LogCluster, a system that clusters system logs to ease log-based problem identification. The framework parses free-form log messages into log events, vectorizes the log events and removes duplicates. Inverted Document Frequency
(IDF) and event weighting is combined prior to clustering, where event weighting is calculated by comparing if a log event occurs in production or lab environment. Agglomerative clustering is applied to the dataset, using cosine similarity as similarity measure. Since 2013, several Microsoft projects have successfully applied LogCluster. The paper evaluates k-means and DBSCAN prior to selecting agglomerative clustering, due to its performance in a distributed environment. The approach is similar to ours, where free-form log messages are being analysed. However, this thesis does not consider whether an error has occurred in production or in a lab environment. There is also a difference in the algorithms that are being applied. The paper evaluates k-means and DBSCAN, but decides to use agglomerative clustering. Our work will not evaluate hierarchical clustering, but rather a combination of density based clustering with hierarchical clustering in addition with x-means which extends k-means.

C. M. Rosenberg and L. Moonen [31] expands this work by studying how hierarchical clustering linkage methods can be improved when used in conjunction with dimensionality reduction. The dimensionality reduction techniques PCA, LSA and non-negative matrix factorization (NMF), are applied to the dataset prior to clustering. The study concludes that log clustering is more accurate when dimensionality reduction is applied, specifically NMF, as it significantly improves the performance of LogCluster. The best performing linkage criteria is complete linkage, which is used in the original paper. The use of dimensionality reduction to improve the work of Q. Lin et al. is interesting, since our work will use some of the techniques that LogCluster applies, in conjunction with dimensionality reduction. However, the clustering algorithm is different, since they are using agglomerative clustering.

### 3.3 Search Strategy

The related works that are presented in this thesis were found by searching for terms relating to the thesis subject in IEEE Xplore and the ACM Digital Library. The search strings used are "log clustering", "test fault clustering", "bug report classification", "test failure clustering", "error log machine learning", "error log classification", "failure clustering", "automated log clustering", and "error log unsupervised learning". The papers were selected based on how similar they were to our research area and topic. The main criteria was that it had to be within the area of unsupervised learning for log analysis or error categorization.
This chapter covers the methodology behind how the data is extracted, pre-processed, and analyzed for reaching the end conclusion.

4.1 Overview

This segment gives a high-level overview of how the study will be conducted. The study is separated into two phases, described in the subsections below.

Phase 1

The purpose of the first phase is to evaluate the clustering performance of HDBSCAN, DBSCAN, and X-means on historical error log data from the last 6 months. Internal validation will be performed for each clustering algorithm, in order to determine which hyperparameter configuration to be used. Thereafter, the clustering results for each model is evaluated by domain experts in order to determine which model outputs the most accurate error clusters for the historical error log data. The accuracy is defined by how well each error is grouped into a cluster of similar errors, and if the root cause behind an error is captured by the clustering model. In addition, historical bug reports from the same time-period is fetched, where the test id’s for each report are matched with the test id’s from the historical error log data. This allows the found clusters to be linked with known bugs, for cases when a bug is found for a clustered error. By having domain-experts evaluate similarities between error logs with bugs, and error logs without bugs in the same cluster; we can determine if root causes are captured by the clustering algorithms. The figure below illustrates every component of the first phase.
4.2 Environment

The purpose of the second phase is to evaluate the outlier detection functionality of HDBSCAN and DBSCAN against the LSTM-based autoencoder. The detected outliers from each model will be evaluated by domain experts, in order to determine which model is most accurate in detecting actual outliers.

Phase 2

This chapter presents the software and hardware environment of this thesis. The software segment gives a brief overview of the different libraries and platforms used in this thesis, as well as a description of which were functions imported for this project.
Software

Python
The Miniconda3 distribution with python version 3.8 was installed. All implementation was made in this environment.

Scikit-learn
The Scikit-learn library provides efficient tools for a wide range of different data analysis applications. Including classification, regression, clustering, dimensionality reduction, and pre-processing.  

Several of the algorithms and methods used for this thesis were enabled by the Scikit-learn library.

- The DBSCAN clustering algorithm is imported from sklearn.cluster, and is used for implementing DBSCAN.
- The TF-idf Vectorizer is imported from sklearn.feature_extraction.text, for applying TF-idf on the error log data.
- The t-SNE dimensionality reduction model is imported from sklearn.manifold, for reducing and forming a visual representation of the high-dimensional error log data.
- The silhouette score metric is imported from from sklearn.metrics, for internally validating the results from different configurations of X-Means.

Hdbscan
The HDBSCAN algorithm from the scikit-learn-contrib/hdbscan git-repository is used for implementing the HDBSCAN algorithm into the python environment. It is based on the original paper by R. Campello et al.

Tensorflow
Tensorflow is an end-to-end open source platform for machine learning applications. It offers a vast and comprehensive ecosystem of tools for building and training machine learning models.

Tensorflow version 2.3.0 is installed for the deep learning applications of this project. Its compatibility with Keras allows the LSTM-based autoencoder to be implemented.

Keras
Keras is an API for deep learning in Python. It runs on top of the tensorflow 2 platform, and provides the essential tools for building efficient deep learning solutions in Python.

The library is used for building the LSTM-based autoencoder.

- The Sequential model is imported from keras.models, as it allows a sequential RNN model to be built.
- The LSTM, Dense, RepeatVector, and TimeDistributed layer is imported from keras.layers.
Pyclustering

Pyclustering is an open-source datamining library written in C++ and Python. It has a focus on cluster analysis applications, and features several algorithms that are not included in Scikit-learn. This includes the X-Means algorithm used in this thesis. [36]

- The X-Means algorithm is imported from pyclustering.cluster.
- The kmeans++ initializer is imported from pyclustering.cluster.center_initializer.

Natural Language Toolkit

Natural Language Toolkit (NLTK) is an open-source platform for natural language programming in Python. It provides a wide range of different text processing libraries, including Porter’s stemming algorithm used in this thesis. [37]

- The Porter Stemmer algorithm is imported from nltk.stem.porter.

Hardware

All implementation is made on machines with the following specification:

1. CPU: Intel(R) Core (TM) i7-10850H CPU @ 2.70GHz
2. RAM: 32 GB
3. GPU: NVIDIA Quadro T1000

4.3 Dataset

All of the data is fetched from ABB Robotics’ internal database for robotic systems tests from the last 180 days, between October 2020 and April 2021. Each data point in the table below represents a result from a test point. Every test point is contained in a set of test points referred to as a test case. The test cases are associated with test runs, which are executed on a nightly basis. The table and text below, describes each column and the overall structure of the data contained in them. The number of rows is 23865, and represents the number of test points.

Table 4.1: Structure of the error data file

<table>
<thead>
<tr>
<th></th>
<th>Error Message</th>
<th>Vera Log</th>
<th>Text File</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23864</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

1. The Error Message column contains a specific error message from a given test point. This error message is a string, and is the last outputted error from a given test point.

2. The Vera Log is an internal log file unique for each test point. The file is built like an XML and contains system information and a description of the build, as well as several log entries. Each log entry describes an event occurring during a test point with a message type, status code, title and a description.

3. The Text file describes all actions the machine executes before a failure occurs in a test. Each row in a Text file consists of date, time, test name, action and a description of the action. It contains three action types; INFO, DEBUG and ERROR.
4.4 Pre-processing

This segment covers how the pre-processing methods were applied for the dataset described in 4.3.

Text Cleaning

This segment describes how the information is cleaned for every column in the dataset. The Porter’s Stemming Algorithm from the \textit{nltk} library \cite{37} is used for the stemming process.

Error message

The following cleaning procedure is conducted for every error message contained in the data.

1. Remove all special characters and digits.
2. Transform the text into lower case.
3. Vectorize the text into a set of tokens, where every token represents a word.
4. Apply Porter’s Stemming Algorithm on every word in the vector.
5. Combine the stemmed words into a new string.

Vera Log

The following cleaning procedure is conducted for every Vera Log contained in the data.

1. Extract only the sequence of status codes for every test point.
2. Save this sequence as a string.

Text File

The following cleaning procedure is conducted for every Text file contained in the data.

1. Extract only unique DEBUG outputs, together with all eventual ERROR outputs.
2. Combine the extracted information into a single string.
3. Vectorize the string into a set of tokens, where every token represents a word.
4. Apply Porter’s Stemming Algorithm on every word in the vector.

Combined Dataframe

After the cleaning process has been performed for every column in the dataset, a dataframe is created with the following structure, as seen in the table below. Every data point is a combination of the cleaned Error message, VeraLog, and TextFile, as described in the above sections.

\begin{table}[h]
\centering
\caption{Structure of the combined dataframe}
\begin{tabular}{ll}
\hline
 & Combined \\
0 & ... \\
1 & ... \\
... & ... \\
23864 & ... \\
\hline
\end{tabular}
\end{table}
4.4. Pre-processing

**TF-idf**

TF-idf is performed on the combined dataframe described in 4.4 resulting in a vector space representation of word frequency for every row in the combined dataframe. The resulting matrix will be used as input for Truncated SVD as described in 4.4. The TF-idf matrix has the following structure.

<table>
<thead>
<tr>
<th></th>
<th>Word 0</th>
<th>Word 1</th>
<th>...</th>
<th>Word 2300</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Word 1 frequency</td>
<td>Word 2 frequency</td>
<td>...</td>
<td>Word 2300 frequency</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23864</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Each column represents a unique word in the documents contained in the combined dataframe, where the total number of columns is equal to 2300, corresponding to the number of unique words. Each row represent a document, which is the combination of a cleaned error message, VeraLog, TextFile, and ConsoleLog, for a particular test point. Every datapoint signifies the TF-idf value for a unique word in a document.

**Truncated SVD**

Since the resulting TF-idf matrix is high dimensional and sparse, Truncated SVD is performed on the matrix. The purpose is to eliminate noise and reduce dimensionality, while keeping a high explained variance ratio in order to minimize potential information loss. The Truncated SVD function from the scikit-learn library is used \[38\]. Several tests are conducted, where the parameter \( n_{\text{components}} \) is iteratively increased. The purpose is to find a configuration which reduces the dimensions sufficiently well, while retaining a sufficiently high explained variance ratio.

<table>
<thead>
<tr>
<th>Components</th>
<th>Explained Variance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.925</td>
</tr>
<tr>
<td>225</td>
<td>0.934</td>
</tr>
<tr>
<td>250</td>
<td>0.942</td>
</tr>
<tr>
<td>275</td>
<td>0.949</td>
</tr>
<tr>
<td><strong>285</strong></td>
<td><strong>0.950</strong></td>
</tr>
<tr>
<td>300</td>
<td>0.954</td>
</tr>
</tbody>
</table>

After conducting the tests, \( n_{\text{components}} = 285 \) is selected since 95 % of the variance is retained. By applying Truncated SVD, the matrix dimensions are decreased from \(23865 \times 2300\) to \(23865 \times 285\).

**Data visualization**

**t-SNE**

The figure below illustrates a scatter plot of the pre-processed data before Truncated SVD is applied, when reduced into three dimensions using t-SNE. In terms of t-SNE hyperparameters, perplexity is set to 20 with 1000 iterations. The coloring scheme is based on the clusters from HDBSCAN with \( \text{min cluster size} = 2 \), and \( \text{min samples} = 2 \). It is used for giving a visual representation of different cluster regions in the scatter plot. The HDBSCAN algorithm uses the pre-processed data with TF-idf as input.
4.4. Pre-processing

Figure 4.3: t-SNE visualization of the data

Truncated SVD

The figure below illustrates a scatter plot of the pre-processed data, when reduced into three dimensions with Truncated SVD (n_components = 3). The coloring scheme is based on the clusters from HDBSCAN with min cluster size = 2, and min samples = 2. It is used for giving a visual representation of different cluster regions in the scatter plot. For this case, the data is pre-processed with all the steps described in 4.4.

Figure 4.4: Visualization of the data with Truncated SVD
4.5 Implementation

This segment covers how the clustering algorithms were implemented, along with how hyperparameters were selected for each algorithm. It also explains how outlier detection was implemented.

Clustering with DBSCAN

The DBSCAN algorithm from the scikit-learn library is used [39]. The error log data is preprocessed as described in 4.4 and used as input for the DBSCAN algorithm. In order to determine which combination of minimum samples and epsilon to use; several configurations of the hyperparameters are benchmarked using the S_Dbw score. In addition, the euclidean and cosine distance metric are evaluated for each configuration. Ester et al. [15] provide a heuristic for selecting the epsilon parameter based on k-nearest neighbour distances. The idea is to compute the average distances of every point to its k-nearest neighbors, where the value k corresponds to minimum samples. This is used as a directive for selecting which range of epsilon for each configuration of minimum samples to be used in the benchmarking process.

![Figure 4.5: Illustration k-nearest neighbor graph](image)

The figure above is an illustration of a k-nearest neighbor graph, where the dashed horizontal line is at the optimal epsilon value. When the optimal configuration has been selected from the benchmarking process, the clustering results from this configuration are evaluated by domain experts, as described in 4.7.

Clustering with HDBSCAN

The error log data is pre-processed as described in 4.4 and used as input for the HDBSCAN model. The python library hdbscan [40] is used for implementing the solution. For HDBSCAN, different values for minimum cluster size and minimum samples are benchmarked with the relative validity score. The relative validity score is a fast approximation of the DBCV score, where the only difference is that relative validity is computed using the mutual-reachability spanning tree, instead of the all-point minimum spanning tree [40]. The configuration which yields the highest relative validity score is evaluated by domain experts, as described in 4.7.
4.5. Implementation

Clustering with X-Means

The error log data is pre-processed as described in 4.4 and used as input for the X-Means algorithm. The python library PyClustering [41] is used for implementing the solution. Since X-Means performs an automatic parameter selection for a range of $k$ using the BIC criteria; several ranges for $k$ are evaluated, where the selected $k$ from X-means is benchmarked with the silhouette score. In addition, the kmeans++ algorithm from the pyclustering library [42] is used for determining the optimal initial centers for a given range of clusters.

Outlier Detection

HDBSCAN and DBSCAN

Both the DBSCAN and the HDBSCAN algorithm automatically assign outliers to data points not fulfilling the requirements of being part of a cluster. If such is the case, both algorithms assign the cluster value of -1 for each data point that is identified as an outlier. The resulting outliers from each algorithm with optimized parameters are sent to domain experts for external evaluation.

LSTM Autoencoder

The error log data is pre-processed as described in 4.4 and used as input for training the LSTM autoencoder. The autoencoder is built and trained with Tensorflow version 2.3.0 [34], and by utilizing the Dense, RepeatVector, and TimeDistributed layer from the Keras library [35]. The architecture of the network is built as follows

```python
# Encoder
lstm_autoencoder = Sequential()
lstm_autoencoder.add(LSTM(128, activation='relu',
                         input_shape=(timesteps, n_features),
                         return_sequences=True))
lstm_autoencoder.add(LSTM(64, activation='relu',
                           return_sequences=False))
lstm_autoencoder.add(RepeatVector(timesteps))
# Decoder
lstm_autoencoder.add(LSTM(64, activation='relu',
                           return_sequences=True))
lstm_autoencoder.add(LSTM(128, activation='relu',
                           return_sequences=True))
lstm_autoencoder.add(TimeDistributed(Dense(n_features)))
```

As seen in the code snippet above, the first LSTM layer consists of 128 units, followed by a second layer of 64 units. The third layer is a repeat vector with the number of units equal to the number of rows in the data-set. Thereafter, two additional LSTM layers consisting of 64 and 128 units are added, in order for the decoder to work. The final layer is a time distributed dense layer, with units equal to the number of features in the data-set. The model is compiled with the Adam optimizer, and uses the mean squared error (mse) as loss function. The autoencoder is trained on 1000 epochs, with a batch size of 64, and a learning rate equal to 0.0001.
4.6 Bug Matching

The reconstruction error (loss) from attempting to reconstruct each data point of the input data, is used as a baseline for classifying a given point as an outlier. The threshold for this loss, is going to be based on the average loss yielded after reconstruction.

4.6. Bug Matching

All historical bug reports from the last 6 months are fetched from ABB’s database. Thereafter, the test point ID, test case ID, and test run ID are matched with the corresponding ID’s in the error log data. This results in a data frame with the following structure

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Vera Log</th>
<th>Text File</th>
<th>Cluster</th>
<th>Bug ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>...</td>
<td>0</td>
<td>1234</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>0</td>
<td>no bug</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
<td>1</td>
<td>no bug</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
<td>1</td>
<td>4351</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23864</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The data frame is thereafter exported as a CSV, and sent to the domain experts for external evaluation. This allows them to analyse if the clustering algorithms manage to capture underlying information that could be caused by a certain bug.

4.7 External Evaluation

The external evaluation is conducted by two domain experts with experience of robotic testing and software development at ABB Robotics. They are tasked with analyzing and evaluating the resulting datasets produced from error categorization, bug matching and outlier detection, which is described in the segments below. The domain experts work together when analyzing and evaluating the results. The authors of this thesis will not be present during the external evaluation.

Evaluation of the Clustering Algorithms

By using expert domain knowledge, the performance of the algorithms can be assessed and compared. The results are given in the form presented in the table below.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Vera Log</th>
<th>Text File</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23864</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

For each of the three clustering algorithms, such a table is produced, sorted by cluster value in ascending order. Since HDBSCAN and DBSCAN has their own outlier detection function, their outliers are marked with a cluster value of -1.
For each table, general information about the clusters are given to help facilitate the comparison of the algorithms. This includes the number of clusters, the average cluster size, and the size of the largest cluster. For HDBSCAN and DBSCAN, the number of outliers are given as well. By using this information, the domain experts conduct an overall inspection of how the clusters are divided. Thereafter, a detailed analysis is made for a sample of three small, three medium sized, and three large clusters. These nine clusters are evaluated based on how useful they are in capturing an error category. After the evaluation has been conducted for each of the three algorithms, the domain experts provide a motivation and description for which one is best suited for the task.

**Bug Matching**

In order to determine how well the bugs are represented in each cluster, the domain experts evaluate the resulting CSV for each clustering algorithm with the following criteria:

- How well does the identified cluster capture the root cause behind a bug?
- Are the clusters linked with the bugs in a sufficient manner?
- If a cluster contains at least one bug; does there exist instances where the non-bug data points in that particular cluster, are non-reported bugs?
- Could this clustering algorithm be helpful in the bug-reporting process?

Similarly to 4.7, the domain experts first conduct a high-level inspection of the clusters containing bugs. Thereafter, a sample of five such bugs are closely evaluated with the criteria above. After the evaluation has been conducted for each of the three algorithms, the domain experts provide a motivation and description for which one is best suited for the task.

**Outlier Detection**

In order to evaluate how well the HDBSCAN, DBSCAN, and LSTM autoencoder detect outliers, a combined CSV containing the results from each solution is sent to the domain experts for evaluation. It is evaluated based on the following criteria:

- Are the detected outliers actual uncommon or abnormal errors, when put in relation to the non-outliers in the data-set?
- Could this solution be a helpful tool for automatically detecting outliers?

The CSV has the following structure:

<table>
<thead>
<tr>
<th>Error Message, Vera Log, Text File</th>
<th>LSTM</th>
<th>HDBSCAN</th>
<th>DBSCAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>Not outlier</td>
<td>Is outlier</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
<td>Not outlier</td>
<td>Not outlier</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23864</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
This chapter covers the results obtained from hyperparameter selection for DBSCAN, HDBSCAN, and X-means through internal validation. Lastly, the chapter presents the results from the external validation, performed by domain experts.

5.1 Implementation

Clustering with DBSCAN

As seen in figure 5.1, several tests for values of $k$ was conducted. $k = [3, 4, 5, 10]$ yielded interpretable results, where an epsilon value between 0.1 and 0.4 is at the elbow for $k = [3, 4, 5, 10]$.

Figure 5.1: K-nearest neighbour graph
By observing the results from the figure above, a parameter-range for \( \text{epsilon} \) and Minimum samples could be selected for internal validation. In the table below, the \( \text{epsilon} \) parameter is adjusted with increments of 0.05 for every value of Minimum samples. As seen in table 5.1, Minimum samples = 3 and Minimum samples = 4 with Epsilon = 0.1 yields the lowest S_Dbw score of 0.021.

Table 5.1: S_Dbw for DBSCAN configurations using euclidean distance.

<table>
<thead>
<tr>
<th>Min samples</th>
<th>Epsilon</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
<th>0.35</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.021</td>
<td>0.037</td>
<td>0.056</td>
<td>0.077</td>
<td>0.097</td>
<td>0.117</td>
<td>0.134</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.021</td>
<td>0.037</td>
<td>0.057</td>
<td>0.079</td>
<td>0.099</td>
<td>0.122</td>
<td>0.139</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.024</td>
<td>0.042</td>
<td>0.062</td>
<td>0.086</td>
<td>0.108</td>
<td>0.134</td>
<td>0.151</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.028</td>
<td>0.047</td>
<td>0.067</td>
<td>0.092</td>
<td>0.114</td>
<td>0.144</td>
<td>0.165</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: S_Dbw for DBSCAN configurations using cosine distance.

<table>
<thead>
<tr>
<th>Min samples</th>
<th>Epsilon</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
<th>0.35</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.141</td>
<td>0.167</td>
<td>0.179</td>
<td>0.182</td>
<td>0.195</td>
<td>0.187</td>
<td>0.157</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.146</td>
<td>0.172</td>
<td>0.181</td>
<td>0.188</td>
<td>0.197</td>
<td>0.184</td>
<td>0.159</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.159</td>
<td>0.192</td>
<td>0.210</td>
<td>0.224</td>
<td>0.246</td>
<td>0.225</td>
<td>0.171</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.174</td>
<td>0.212</td>
<td>0.237</td>
<td>0.248</td>
<td>0.272</td>
<td>0.267</td>
<td>0.233</td>
<td></td>
</tr>
</tbody>
</table>

As seen in table 5.2, the configuration with Minimum samples = 3 and \( \text{epsilon} \) = 0.1 yields the lowest S_Dbw score of 0.141.

In addition to the above tests, an evaluation was made for Minimum samples = 285 and Minimum samples = 570, where \( \text{epsilon} \) was evaluated in the range \([1, 1.2]\). These configurations resulted in one or zero clusters. The resulting clusters from the two configurations with Minimum samples = 3 and Minimum samples = 4 where Epsilon = 0.1 from table 5.1 are sent to domain experts for external evaluation.

Clustering with HDBSCAN

The following hyperparameter configurations are evaluated. As seen in table 5.3 and 5.4, min cluster size remains unchanged. This is due to the nature of the data, coupled with the desire of having datapoints to be marked as an outlier if they are not similar to any other datapoint. The configuration with min cluster size = 2, and min samples = 2 with the euclidean distance metric yields the highest relative validity score. The resulting clusters from this configuration is sent to the domain experts for external evaluation.

Table 5.3: HDBSCAN configurations with euclidean distance.

<table>
<thead>
<tr>
<th>min cluster size</th>
<th>min samples</th>
<th>distance metric</th>
<th>relative validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>Euclidean</td>
<td>0.693</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Euclidean</td>
<td>0.612</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>Euclidean</td>
<td>0.590</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Euclidean</td>
<td>0.538</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Euclidean</td>
<td>0.502</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>Euclidean</td>
<td>0.463</td>
</tr>
</tbody>
</table>
Table 5.4: HDBSCAN configurations with an approximation of the cosine distance.

<table>
<thead>
<tr>
<th>min cluster size</th>
<th>min samples</th>
<th>distance metric</th>
<th>relative validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>Cosine</td>
<td>0.661</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Cosine</td>
<td>0.591</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>Cosine</td>
<td>0.581</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Cosine</td>
<td>0.509</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Cosine</td>
<td>0.465</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>Cosine</td>
<td>0.408</td>
</tr>
</tbody>
</table>

**Clustering with X-means**

The following cluster ranges are evaluated. As seen in table 5.5, number of clusters = 2400 yield the highest silhouette score of 0.763. The resulting clusters from this configuration is therefore sent to the domain experts for external evaluation.

Table 5.5: X-Means configurations

<table>
<thead>
<tr>
<th>cluster range</th>
<th>selected number of clusters</th>
<th>silhouette score</th>
</tr>
</thead>
<tbody>
<tr>
<td>[500, 1000]</td>
<td>1000</td>
<td>0.678</td>
</tr>
<tr>
<td>[1500, 1800]</td>
<td>1800</td>
<td>0.751</td>
</tr>
<tr>
<td>[1800, 2000]</td>
<td>2000</td>
<td>0.757</td>
</tr>
<tr>
<td>[2000, 2400]</td>
<td>2400</td>
<td>0.763</td>
</tr>
</tbody>
</table>

**Outlier Detection**

**HDBSCAN and DBSCAN**

The HDBSCAN algorithm with minimum cluster size = 2, and minimum samples = 2 is used for outlier detection. 1668 outliers are detected out of 23864 samples.

For DBSCAN, the configuration with epsilon = 0.1, and minimum samples = 3 is used for outlier detection. 3123 outliers are detected out of 23864 samples.

**LSTM autoencoder**

During training, an exponential decay can be observed for the loss in figure 5.2 in addition with convergence after 600 epochs.
5.1. Implementation

The resulting reconstruction errors can be observed in Figure 5.2. The x-axis represents each sample in the data-set, where the y-axis is the computed loss between the reconstructed and actual data points. The mean value for all losses is 0.000236. The threshold for outliers is defined as all samples with losses equal to or above 0.0025. This leads to 100 identified outliers out of 23864 samples.

Figure 5.2: Average reconstruction loss per epoch

![Model loss graph](image)

The x-axis represents each sample in the data-set, where the y-axis is the computed loss between the reconstructed and actual data points. The mean value for all losses is 0.000236. The threshold for outliers is defined as all samples with losses equal to or above 0.0025. This leads to 100 identified outliers out of 23864 samples.

Figure 5.3: Reconstruction loss per sample

![Reconstruction loss graph](image)
5.2 **External evaluation**

Below, the results from the external evaluation are presented. This section is divided into three subsections; the algorithms ability to cluster similar failure causes, their ability to capture information pertaining to bugs, and outlier detection performance when compared to an LSTM autoencoder. For each section, the domain experts rank the methods based on their analysis.

**Evaluation of the Clustering Algorithms**

In this subsection, the algorithms ability to cluster similar failure causes are analyzed and ranked by the domain experts. This is conducted by evaluating how well clusters of different sizes represent an error category, and if they are accurate enough.

**DBSCAN**

DBSCAN produced 1360 clusters, with an average cluster size of 15.25 data points. 3123 outliers were detected. According to the domain experts, too many outliers are identified by the algorithm. As many of the outliers were common errors which should belong to a cluster. Furthermore, they state that the average cluster size is too large when put in relation to the amount of data that is fetched coupled with the types of robotic tests that were conducted during that time period.

- Small-sized clusters with 4-13 data points gives a good representation of different error categories. Many of the small clusters capture error categories very well. Generally good separation between clusters. Although, there exist some instances where different categories occur within the same cluster.

- Medium-sized clusters of size 21-50 data points gives a tolerable representation of different error categories. There are instances where different clusters represent the same error-category.

- Large-sized clusters of size 97-688 data points gives a tolerable representation of different error categories. There are instances of several error categories being present in one cluster. The separability between clusters is okay, even though there are many instances where the clusters are well-defined.

**HDBSCAN**

HDBSCAN produced 2385 clusters with an average cluster size of 9.31 data points. 1668 outliers were detected. According to the domain experts, the average cluster size is a reasonable number, given the amount of data that is fetched coupled with the types of robotic tests that were conducted during that time period.

- Small-sized clusters with 3-6 data points has a tolerable accuracy. However, several clusters of the same error category exist, where they instead should be one cluster.

- Medium-sized clusters with 13-19 data points has a tolerable accuracy. There exist many instances where different clusters represent the same error-category. In addition, some instances of miscategorized errors are observed.

- Large-sized clusters with 100-400 data points performs well overall. Several instances of well-captured categories are observed. Although, there are certain instances where multiple clusters of the same category exist.
5.2. External evaluation

X-means

X-Means produced 2400 clusters with an average cluster size of 10 data points, and is according to the domain experts; a reasonable number, given the amount of data that is fetched coupled with the types of robotic tests that were conducted during that time period.

- Small-sized clusters with 2-4 data points have clearly separated clusters. Data points in the same cluster are similar to each other, and clearly different from data points in other clusters.

- Medium-sized clusters of size 16-29 data points have clearly separated clusters. Data points in the same cluster are similar to each other, and different to data points in other clusters.

- Large-sized clusters of size 100-1143 data points are overall good, with clearly separated clusters. The largest cluster of 1143 data points is approximately 90% correct.

Ranking

Based on the above analysis, the domain experts ranked the algorithms as follows:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X-Means</td>
</tr>
<tr>
<td>2</td>
<td>HDBSCAN</td>
</tr>
<tr>
<td>3</td>
<td>DBSCAN</td>
</tr>
</tbody>
</table>

They found X-Means to be most consistent and accurate for the task of categorizing error types through clustering.

Bug Matching

In this subsection, the domain experts analyze and rank the clustering algorithms based on their ability to capture information pertaining to known bugs, when producing the clusters.

DBSCAN

- **How well does the identified cluster capture the root cause behind a bug?**
  
  There are instances where specific clusters seem to capture the root cause behind a bug. This is especially apparent in four out of the five bugs that were analyzed. Where error logs that were not marked as a bug were strikingly similar to the error log that was, and where these similar logs were inside the same cluster. This suggests that the algorithm was able to capture such information. However, with regards to the fifth bug, there are around 100 instances of it being tied to data points that are identified as outliers, even though they shouldn’t be. This particular bug was also found inside other clusters, which is incorrect.

- **Are the clusters linked with the bugs in a sufficient manner?**
  
  Although there are instances where information pertaining to certain bugs are captured. The overall inconsistencies relating to a high number of misidentified outliers and instances of miscategorized error types, lead to the conclusion that it is insufficient in bug matching.

- **If a cluster contains at least one bug; does there exist instances where the non-bug data points in that particular cluster, are non-reported bugs?**
Such instances do exist. However, there are also instances where one bug is found in several different clusters and in data points marked as outliers. This makes the performance with regards to bug matching, inconsistent and unreliable.

- **Could this clustering algorithm be helpful in the bug reporting process?**
  It could be helpful in bug-reporting for specific cases. However, since the clustering results are overall too inconsistent, it is not believed to be sufficient enough.

**HDBSCAN**

- **How well does the identified cluster capture the root cause behind a bug?**
  HDBSCAN seems to be able to accurately capture the root cause behind four out of the five bugs that were analyzed. In these cases, error logs that were previously not marked as a bug, could now be accurately classified as such. However, the fifth bug was found in several different clusters. Overall, HDBSCAN seems to be able to capture the root cause behind certain bug types.

- **Are the clusters linked with the bugs in a sufficient manner?**
  In some ways, it is. But when considering that there exist multiple clusters within the same error category, which in turn, could result in inconsistent bug matching results. It is concluded that the algorithm is insufficient for the task.

- **If a cluster contains at least one bug; does there exist instances where the non-bug data points in that particular cluster, are non-reported bugs?**
  Yes. This could clearly be seen in four out of the five analyzed bugs. For these cases, the algorithm provided good assistance in detecting such instances.

- **Could this clustering algorithm be helpful in the bug reporting process?**
  Overall, we believe that it could. However, it would most likely only work for certain bug types.

**X-Means**

- **How well does the identified cluster capture the root cause behind a bug?**
  X-Means was able to capture the root cause behind all of the five bugs that were analyzed. The fifth bug, which proved difficult for HDBSCAN and DBSCAN to capture, was sufficiently captured by X-Means with only minor inconsistencies.

- **Are the clusters linked with the bugs in a sufficient manner?**
  Yes. The clusters were matched in a sufficient manner, with minor inconsistencies.

- **If a cluster contains at least one bug; does there exist instances where the non-bug data points in that particular cluster, are non-reported bugs?**
  Yes. For all of the five bugs that were analyzed, X-Means yielded the strongest results, as it successfully identified previously non-reported bugs.

- **Could this clustering algorithm be helpful in the bug reporting process?**
  Yes. Since X-Means is very consistent in categorizing error types, in conjunction with capturing bugs associated with the identified clusters. It is believed that X-Means has the potential of being very helpful in the bug reporting process.

**Ranking**

Based on the above analysis from the domain experts, they ranked the algorithms as follows:
5.2. External evaluation

Table 5.7: Algorithm rank: bug matching performance

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X-Means</td>
</tr>
<tr>
<td>2</td>
<td>HDBSCAN</td>
</tr>
<tr>
<td>3</td>
<td>DBSCAN</td>
</tr>
</tbody>
</table>

Outlier Detection

In this subsection, the domain experts analyze and evaluate the selected outlier detection methods. The last part of this subsection includes additional findings, followed by a ranking of the different methods.

LSTM-Autoencoder

- Are the detected outliers actual uncommon or abnormal errors, when put in relation to the non-outliers in the data-set?
  The instances with highest reconstruction error do not represent outliers particularly well. The outliers identified by the algorithm are also similar to each other and should probably be in a cluster. In addition, none of the data points above the threshold are actual outliers.

- Could this solution be a helpful tool for automatically detecting outliers?
  No.

HDBSCAN

- Are the detected outliers actual uncommon or abnormal errors, when put in relation to the non-outliers in the data-set?
  HDBSCAN detected 1668 outliers in the dataset. Certain instances do, in fact, represent uncommon errors. If the algorithm would predict on new data, there is a fair chance that uncommon instances would be marked as outliers. However, the number of outliers is very high, where many them should have been assigned to a cluster.

- Could this solution be a helpful tool for automatically detecting outliers?
  It would be helpful. But the results can’t be entirely relied upon.

DBSCAN

- Are the detected outliers actual uncommon or abnormal errors, when put in relation to the non-outliers in the data-set?
  DBSCAN detected 3123 outliers. The high number of outliers indicate that it would not be practical to utilize DBSCAN as an outlier detection method for the data. Many of the found outliers should be within a cluster.

- Could this solution be a helpful tool for automatically detecting outliers?
  No, the results are too inconsistent.

Additional Findings

Although X-Means does not detect outliers, it clusters error logs with no other similar log into a cluster with a size = 1. Such instances were found by the domain experts to yield the best representation of an uncommon error log. Furthermore, they found that this approach is the most consistent and accurate, when compared with the other outlier detection results.
5.2. External evaluation

Ranking

Based on the above analysis from the domain experts, they ranked the algorithms as follows:

Table 5.8: Ranking: Outlier detection

<table>
<thead>
<tr>
<th>Rank</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X-Means</td>
</tr>
<tr>
<td>2</td>
<td>HDBSCAN</td>
</tr>
<tr>
<td>3</td>
<td>DBSCAN</td>
</tr>
<tr>
<td>4</td>
<td>LSTM autoencoder</td>
</tr>
</tbody>
</table>
In this chapter the results, method and the work in a wider context will be discussed.

6.1 Results

Internal validation

The results from [5.1] suggest that the best parameter configuration for DBSCAN is *Minimum samples* = 3 or 4, with an *Epsilon* value of 0.1. When observing table [5.1] and [5.2] one can see that the S_dbw score improves as the parameter values are lowered. This is a result of the validation metric favoring a higher cluster count, as this leads to smaller, but more distinct clusters. In order to yield valuable and interpretable results, the k-nearest neighbour graph was used for determining the parameter-range prior to internal validation, as it is a recommended parameter selection method by the original authors of DBSCAN.

For HDBSCAN, the results in table [5.3] and [5.4] suggest that *min cluster size* = 2 and *min samples* = 2 yields the best clustering results. This could be due to the nature of the error log data, as there are instances where at least two identical error logs occur from different test points. If *minimum samples* is above 2, some of these instances would be identified as noise. This leads to less distinct dense regions, which in turn would lower the DBCV score, which can be observed in the tables.

With regards to distance measures, both cosine similarity and euclidean distance were validated for HDBSCAN and DBSCAN. When using cosine similarity, fewer clusters and outliers were found for both algorithms, compared to when euclidean distance was applied. This resulted in slightly worse validity scores, which do not necessarily reflect how well different error categories are captured. Although, upon further inspection of resulting clusters from both distance measures, cosine similarity had cases of distinctly different points appearing within the same cluster. This was also observed by the domain experts, when early results were sent to them prior to the formal external evaluation described in [5.2]. The reason for this, was to mitigate the issue of solely relying on the internal validation methods, since it could result in an unfair comparison between HDBSCAN and DBSCAN. Especially when
considering that DBSCAN requires more parameters to be tuned compared to HDBSCAN.

For X-means, the results in table 5.5 indicate that the best performing cluster range is [2000, 2400] resulting in 2400 clusters. In addition to X-Means’ internal BIC score, the silhouette score improved as the clusters grew. It can be argued that the optimal amount of clusters could potentially be above 2400. However, the high cluster count is due to X-Means not being able to detect noise, where it will instead construct clusters containing one data point. Such instances are not necessarily desirable, as the average cluster size would decrease, and in turn yield clusters containing only identical data points. Since error categories could contain non-identical data points, albeit still similar, it was decided to settle on a cluster size of 2400.

### Choices made regarding the LSTM autoencoder

As seen in figure 5.3, the mean squared error (MSE) for each reconstructed point is below 0.0035. This indicates that the autoencoder is able to reconstruct the data points with a very high accuracy. The choice of selecting a reconstruction error threshold of 0.0025 was made under the assumption that if a data point has a very high reconstruction error relative to the other data points, it is very likely to be unique. With this threshold, the top 100 data points with the highest reconstruction error are marked as outliers. This made it easier for the domain experts to determine if the solution is viable or not, since these data points are more likely to stand out in the data. When interpreting figure 5.2 one can observe a convergence of the average loss after 600 epochs. Because of this, a decision was made to set the number of epochs equal to 1000 when training the model. It could be argued that such a high epoch count could lead to the model over-training, but since the objective is to find outliers from reconstructions with high losses, this was not the greatest of concerns. Especially since the data points that were harder to reconstruct always had a higher loss relative to the other data points after around 300 epochs.

### External Evaluation

#### Evaluation of the Clustering Algorithms

As seen in table 5.6, the domain experts considered X-means to be the best algorithm for error type categorization. This was followed by HDBSCAN, which produced somewhat inconsistent results where certain error types were found in several clusters. This issue was even more prominent with DBSCAN. These results were initially surprising, as DBSCAN and HDSCAN performs well on arbitrary data shapes, which was expected for high-dimensional text data that is pre-processed and domain-specific. However, it could also be argued that the resulting TF-idf matrix produces cluster regions that are not of an arbitrary shape. This is joined by the fact that the results from the internal validation favours clusters of smaller sizes, which in turn could indicate that arbitrary shapes are not present in the way we initially thought.

#### Bug matching

As can be seen in table 5.7, the domain experts considered X-means to be the best algorithm for capturing information pertaining to known bugs in the clusters. This was followed by HDBSCAN and DBSCAN, which gave inconsistent results. This is not too surprising, when considering that X-Means also was most consistent in categorizing error types. Additionally, X-Means has clearly separated clusters, meaning that the connection to previously known bugs will also be clearly separated since each cluster will only contain one type of error which can be linked to a bug.
6.2 Method

Outlier detection

According to the domain experts, none of the three methods were sufficient in detecting outliers in the data. The LSTM autoencoder was surprisingly insufficient in detecting unusual errors occurring in the data. An interesting observation from these results were that there were certain types of identical errors that had the highest reconstruction error. This could be a result of a weighting problem in the dataset, where the TextFile tends to hold a lot more information than both Error Message and Vera Log. Since LSTMs take sequences as input, data points with more information could be harder to reconstruct. Rather than data points that have different textual information when put in relation to the other data points. The poor results could also be due to TF-idf discarding the order in which words appeared in the original data. Therefore, it could be interesting to investigate other means of feature selection and pre-processing when feeding data into the model. One alternative approach could be to only take sequences of status codes as input data.

However, HDBSCAN had instances where it successfully detected outliers, although it was inconsistent in doing so, as there also were many instances of non-outliers being miscategorized. The same is true for DBSCAN, although it had many more instances of miscategorized outliers. One thing that we did not consider when formulating our research questions, was the possibility that clusters containing only one data point could be viewed as outliers. As the domain experts analyzed the results from X-Means, they found that such instances gave the best representations of outliers appearing in the data. This means that the partitional clustering approach yielded the most favourable results for all of the three research questions in this thesis.

6.2 Method

Pre-processing

There are many different ways in which data can be pre-processed, and greatly depends on the shape of the data, in conjunction with the selected models for analysing the given data. The data used in this thesis are logs with large amounts of text, joined by error messages. Since many abbreviations and domain-specific expressions are present in this data; several state-of-the-art word vectorization methods such as GloVe and Word2Vec had to be excluded. This is due to these methods using pre-trained weights on real words for finding semantic similarities between words in text. Since such models would be ineffective for instances where non-real words frequently occur, TF-IDF was selected for creating a vector representation of the error log data. Where Truncated SVD is thereafter used for dimensionality reduction and potentially reducing noise in the data. Some interesting observations were made when evaluating the effectiveness of applying both TF-IDF and Truncated SVD (LSA) prior to clustering. For HDBSCAN, the relative validity for minimum samples = 2 and minimum cluster size = 2, was higher when LSA was used compared to solely using TF-IDF. This could suggest that LSA removed more noise than information.

However, some potential improvements could be made with regards to how the data was handled prior to clustering. L. Qingwei et al. [2] added weights to the frequency values in the data matrix, based on if an error occurs in production or in a lab environment. A similar approach could have been used here, where a weight scheme based on log file information could have been implemented. For example, the Text File log contains more terms than the Vera Log and Error message. This could lead to skewed results, where if a test point contains a very large Text File it would be unevenly weighted against other test points. Even though this issue is mitigated by only extracting the unique DEBUG outputs as described in [4]. The
issue could still be present, as there might be instances of some Text File logs having a large amount of unique DEBUG outputs.

### Implementation

The choice of using $S_{dbw}$ as the internal validation metric for DBSCAN was based on DBCV’s poor performance on high-dimensional data. Which stayed true even after the dimensions were reduced to $23864 \times 285$ with Truncated SVD. In addition, both $S_{dbw}$ and relative validity tends to have a positive bias toward parameter configurations that yield smaller clusters with identical points. This is joined by the inherent difficulty of handling the resulting outliers from HDBSCAN and DBSCAN, since they are not actual clusters. This is true for all currently available internal validation metrics, since they identify the outliers as clusters. In addition, since there is not a way to compute an accuracy metric for unsupervised learning applications, internal validation is the next best choice for evaluating hyperparameter configurations. The other alternative would be to have domain experts evaluate the results for each individual configuration. This is a more optimal way for selecting hyperparameters, although it requires more resources, and is time consuming. Therefore, a decision was made to use the internal validation metrics for selecting hyperparameters, while having domain experts evaluate each resulting algorithm individually.

The choice of algorithms was motivated by several factors. Since we were dealing with large amounts of textual data with a problem description that required an unsupervised approach; a decision was made to evaluate the performance of clustering algorithms that covered a wide spectrum of clustering techniques. X-Means was selected as it is an extended version of k-Means where the k number of clusters is not needed to be defined, as it is automatically determined through performing centroid splits based on the BIC. Since the number of error categories is undefined, it was especially suitable for the given data. This is also true for HDBSCAN, as it utilizes the strengths of both hierarchical and density-based clustering for finding cluster regions in the dataset. DBSCAN was also selected for this reason, as it is a density-based algorithm in which HDBSCAN is based upon. Another reason for selecting the three algorithms was due to the high-dimensionality of the data once it was pre-processed. Since it is difficult to visualize, it is challenging to select an algorithm based on how the data is plotted.

### External Evaluation

In order to determine how well each algorithm clusters the error log data, a decision was made to have experienced developers with domain expertise perform a comparative evaluation of the resulting clusters from each algorithm. The domain experts have years of experience with interpreting errors from the robotic test systems, and also provided assistance in the process of selecting which log files that would be valuable in capturing different error types. Although this could be viewed as valid approach for selecting which information to include in the dataset, this could also lead to human bias being implicitly contained in the data. Which in turn results in the algorithms clustering subjectively selected information. This was something that was taken into consideration when forming the dataset. Although on the other hand, this could also lead to a more effective feature selection, as it would mitigate the issue of having unnecessary information and noise in the dataset, prior to pre-processing.
6.3 The work in a wider context

Validity Threats

There are several types of validity threats that most quantitative studies should take into consideration when validating results. This includes construct validity, internal validity, external validity, and conclusion validity [43].

Construct validity refers to whether the study measures what was intended or not, which in this case is if error categories can be found by means of clustering, and if it is possible to detect outliers in the process. This is measured by external evaluation, where the results are validated by domain experts who have worked with this data for a long time. Therefore, we believe that we have measured what we intended. Especially when considering the challenging nature of measuring accuracy for unsupervised learning applications. As for internal validity, it refers to the casual relationships between the data and algorithms used in this thesis. We perform the same pre-processing steps prior to using each algorithm, where these steps are widely used within NLP. In addition, we utilized recommended internal validation metrics for selecting hyperparameters for each algorithm, which mitigates the issue of achieving skewed results as an effect of using unoptimized hyperparameters.

External validity concerns the generalizability of our results. Thus, in order to generalize our results to other domains and applications, other datasets will have to be used. Therefore, it is of high importance to construct a dataset of a similar structure as the one defined in the method chapter, in order to achieve similar results. Conclusion validity refers to how reasonable the conclusions we reach are. Since all conclusions are based on the external evaluations from domain experts, we find that our conclusions regarding which of the three algorithms perform the best for the given data, are sound.

Replicability

A full replication of this study is not possible, since the dataset is not available to the public. However, all methods used for achieving the results were enabled by open-source software libraries. Therefore, a similar methodology could be replicated for unsupervised learning with error log data. Although the hyperparameters will most likely have to be tuned, as the corpus and dataset in this study is heavily domain-specific.

Reliability of sources

The sources used in this thesis are from books, academic papers and conference proceedings. Most of the sources have multiple citations and are peer reviewed. In addition, online sources for the documentation of software libraries are cited as well. Therefore, the sources are considered reliable.

6.3 The work in a wider context

This specific thesis is not applicable in a wider context. However, by automating the process of inspecting faults and bugs for software systems; it could potentially enable a more effective development process, where developers can spend more time on developing new components. In addition, this thesis could inspire further work on similar problems or datasets. Enabling further progress to be made in related fields.
Conclusions and future work

7.1 Research Questions

This thesis evaluated the clustering algorithms DBSCAN, HDBSCAN, and X-Means for categorizing error types and capturing bugs. In addition, HDBSCAN, DBSCAN, and an LSTM autoencoder were evaluated for detecting outliers in the error log data. The results from the external evaluation indicate that X-Means outperform both DBSCAN and HDBSCAN when categorizing error types, and when capturing bugs in the clustering process. For outlier detection, neither HDBSCAN nor DBSCAN were able to detect outliers in a sufficiently accurate and consistent manner. In addition, the data points with the highest reconstruction error from training the LSTM autoencoder did not represent outliers. However, additional findings suggest that X-Means’s clusters containing one data point, do provide an accurate representation of outliers in the given dataset.

How does HDBSCAN, DBSCAN, and X-means compare, when tasked with producing clusters that represent different error types?

By observing the results from the external evaluation, it was found that X-Means produce the most accurate representations of different error categories with the given pre-processed error log data. This is followed by HDBSCAN, as it produced sufficiently accurate error categories with the drawback of producing multiple clusters of the same category, which is not practical for real-world applications. As for DBSCAN, this issue was found to be even more prominent, and is joined by DBSCAN’s large number of misidentified outliers, which prevents many errors from being categorized. Therefore, it can be concluded that out of the three algorithms, X-Means provide the most accurate and consistent results. This also implies that k-Means would work equally well, as long as the number of k clusters is optimized through the Bayesian Information Criterion.

Is it possible to draw a connection between found clusters from the error log data with known bugs?

The results from the external evaluation suggest that it is possible, especially when using X-Means. The domain experts observed that bugs were captured in the clusters, as the algo-
algorithm consistently separated bug-related clusters with similar non bug-related clusters. The algorithm was able to make connections that otherwise would require a thorough manual inspection from an experienced developer. When put into comparison with the other algorithms, it can be concluded that X-Means is most suitable for such applications.

Is an RNN-based autoencoder better suited for finding uncommon or abnormal error logs, when compared to the outliers found by DBSCAN and HDBSCAN?

The results from the external evaluation indicate that this is not the case, at least when using the pre-processed error log data as input data. In addition, it was also concluded that HDBSCAN’s outlier detection could potentially be viable for real-world applications, as it did provide promising yet inconsistent results. However, this was not the case for DBSCAN, as it was not accurate in detecting outliers. It was also discovered that the clusters with a size equal to 1 from X-Means, gave the most accurate representation of outliers in the data, and is something that should be explored further.

7.2 Future work

There are areas of this thesis that would be interesting to explore further. One interesting topic for future work are different ways in which the input data can be pre-processed. One option could be to explore how important each log file is, and how these files could be weighted in order to improve results further. Furthermore, different word representation techniques could be tested. For example, techniques that does not discard the order in which words appear in a sentence. This would make use of the order in which status codes and sentences appear in the log files. Although it is tricky to implement for text data that includes domain-specific words and phrases; it could potentially lead to greater improvements in clustering performance.


[42] **Kmeans_plusplus_initializer class reference**: Distributed asynchronous hyper-parameter optimization. [Online]. Available: [https://pyclustering.github.io/docs/0.8.2/html/db/de0/classpyclustering_1_1cluster_1_1center_initializer_1_1lkmeans__plusplus__initializer.html](https://pyclustering.github.io/docs/0.8.2/html/db/de0/classpyclustering_1_1cluster_1_1center_initializer_1_1lkmeans__plusplus__initializer.html) (accessed: 27.04.2021).