Automating debugging through data mining
Automatisering av felsökning genom data mining

JULIA THUN
REBIN KADOURI
Automating debugging through data mining
Automatisering av felsökning genom data mining

Julia Thun
Rebin Kadouri

Examensarbete inom
Datateknik,
Grundnivå, 15 hp
Handledare på KTH: Reine Bergström
Examinator: Ibrahim Orhan
TRITA-STH 2017:23

KTH
Skolan för Teknik och Hälsa
141 57 Huddinge, Sverige
Abstract

Contemporary technological systems generate massive quantities of log messages. These messages can be stored, searched and visualized efficiently using log management and analysis tools. The analysis of log messages offer insights into system behavior such as performance, server status and execution faults in web applications.

iStone AB wants to explore the possibility to automate their debugging process. Since iStone does most parts of their debugging manually, it takes time to find errors within the system. The aim was therefore to find different solutions to reduce the time it takes to debug.

An analysis of log messages within access – and console logs were made, so that the most appropriate data mining techniques for iStone’s system would be chosen. Data mining algorithms and log management and analysis tools were compared. The result of the comparisons showed that the ELK Stack as well as a mixture between Eclat and a hybrid algorithm (Eclat and Apriori) were the most appropriate choices. To demonstrate their feasibility, the ELK Stack and Eclat were implemented. The produced results show that data mining and the use of a platform for log analysis can facilitate and reduce the time it takes to debug.

Keywords
Association rule mining; Machine learning; Classification algorithms; Supervised learning; Text mining; Log management and analysis tools.
Sammanfattning

Dagens system genererar stora mängder av loggmeddelanden. Dessa meddelanden kan effektivt lagras, sökas och visualiseras genom att använda sig av logghanteringsverktyg. Analys av loggmeddelanden ger insikt i systemets beteende såsom prestanda, serverstatus och exekveringsfel som kan uppkomma i webbapplikationer.

iStone AB vill undersöka möjligheten att automatisera felsökning. Eftersom iStone till mestadels utför deras felsökning manuellt så tar det tid att hitta fel inom systemet. Syftet var att därför att finna olika lösningar som reducerar tiden det tar att felsöka.


Nyckelord
Association rule mining; Maskininlärning; Classification algorithms; Supervised learning; Text mining; Logghanteringsverktyg,
Nomenclature

The ELK Stack - Elasticsearch, Logstash and Kibana is a combination of log management components that together makes an end-to-end stack for searching and analyzing data.

SaaS - Software as a Service, the owner of the software shares access to it, often as a subscription through the cloud.

JVM - Java Virtual Machine. Software that enables computers to run Java programs. It translates Java binary code into processor specific instructions.

Wrapper function - Subroutine which often calls another subroutine with some or none additional computation. The java service wrapper launches the JVM.

Apache Lucene - Java text search engine library. Especially suitable for cross-platform.

ARM – Association Rule Mining. Finding frequently upcoming associations within a collection of items.
## Contents

1 **Introduction** ..................................................................................................... 1  
1.1 Problem statement ..................................................................................... 1  
1.2 Purpose ...................................................................................................... 1  
1.3 Scope and limitations ................................................................................. 2  

2 **Background** ..................................................................................................... 3  
2.1 The data mining process ............................................................................ 3  
2.1.1 The preprocessing phase .................................................................. 3  
2.1.2 The analytical phase .......................................................................... 3  
2.1.3 Association rule mining .................................................................... 4  
2.1.4 Machine Learning .............................................................................. 8  
2.1.5 Text mining, term frequency ........................................................... 10  
2.2 Full-text search ......................................................................................... 10  
2.3 Related work ............................................................................................ 11  
2.3.1 Implementation of the ELK Stack ................................................... 11  
2.3.2 Finding and analysing patterns within console logs .................... 12  
2.3.3 Improve effectiveness of searching for information within documents ..................................................................................................... 12  

3 **Methods** ......................................................................................................... 17  
3.1 Overview of the current system ................................................................ 17  
3.2 Choosing the right search engine ............................................................. 17  
3.2.1 Techniques used in Lucene ............................................................ 18  
3.2.2 Techniques used in Sphinx ............................................................. 19  
3.2.3 Comparison of Lucene and Sphinx ................................................ 20  
3.3 Choosing the right log management and analysis tool ............................. 21  
3.3.1 The ELK Stack.................................................................................. 21  
3.3.2 Splunk ............................................................................................... 24  
3.3.3 A comparison of log management and analysis tools ................. 25  
3.4 Implementation of Logstash ..................................................................... 27  
3.5 Implementation of the data mining process .............................................. 29  
3.5.1 The preprocessing phase................................................................ 29  
3.5.2 The analytical phase ........................................................................ 30  

4 **Result** ............................................................................................................. 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 The ELK Stack</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Association rule learning</td>
<td>38</td>
</tr>
<tr>
<td>4.3 Supervised learning</td>
<td>38</td>
</tr>
<tr>
<td>4.4 Text mining, term frequency</td>
<td>40</td>
</tr>
<tr>
<td>5 Analysis and discussion</td>
<td>41</td>
</tr>
<tr>
<td>5.1 The ELK Stack</td>
<td>41</td>
</tr>
<tr>
<td>5.2 ARM and Eclat</td>
<td>42</td>
</tr>
<tr>
<td>5.3 Machine Learning algorithms</td>
<td>42</td>
</tr>
<tr>
<td>5.4 Text Mining</td>
<td>43</td>
</tr>
<tr>
<td>5.5 Social, economic, environmental and ethical aspects</td>
<td>43</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>45</td>
</tr>
<tr>
<td>6.1 Future research</td>
<td>45</td>
</tr>
<tr>
<td>7 Bibliography</td>
<td>47</td>
</tr>
</tbody>
</table>
1 Introduction

iStone [1] was founded in 2007 by Markus Jakobsson whose main goal was to provide optimized solutions within digital commerce and business systems for the company clients. The number of employees at iStone has increased significantly from a middle-sized company with 25 full time employees to more than 500 over the past years, and has expanded nationwide to countries such as Norway, USA and Chile.

The process of debugging within iStone’s own e-commerce system includes analysing log messages generated from their servers in advance to find errors. It is time-consuming to search for errors in each server by analysing log messages. To find the faults that are caused in the system, iStone developers currently need to manually analyse patterns that are causing the problem in the system, this takes time and a great deal of resources. Automating the debugging inside iStone’s system demands less resources, which is likely needed. The contribution of our prototype should not just benefit iStone alone but any company that wishes to enhance their system.

1.1 Problem statement

The vulnerability to an e-commerce shutting down, receiving security threats, or uncompleted purchases is a huge issue. Some problems can reside in the interlocking servers due to poor coding. A server could be overloaded with too many clients causing bad response codes. Nevertheless, techniques such as data mining algorithms, regex and full text search can be used in various tools which in turn help isolate data from the database. Implementing a tool that enables automatic debugging, among the interacting systems, could help reduce the resources needed.

Currently, iStone does manual debugging whenever something goes wrong in the e-commerce process, which is incredibly time-consuming. This thesis explores the possibilities of reducing the time of debugging, by making this process automatic through log management.

1.2 Purpose

The aim of our research is to analyse erroneous data from server logs, data mine it and present the result in a visualization dashboard. This thesis is divided into several objectives:

1. Research regarding automatic debugging
   a. Find and compare different log management tools to decide which one to implement.
   b. Gather information about different data mining algorithms used for pattern recognition.

2. Create a test environment and implement a prototype
   a. Implement the chosen log management tool in the test environment.
   b. Test the tool by collecting and visualize the access - and console logs.
c. Implement data mining algorithms on the collected log messages.
   d. Analyze the result.

1.3 Scope and limitations

- Time limit for the thesis work is 10 weeks.
- This study will focus on access- and console logs generated during 5 days from one of iStone’s customers.
2 Background

This chapter consists of 3 sections. Section 2.1 includes an introduction to the data mining process and includes descriptions of the different data mining techniques: association rule mining, machine learning and text mining.

Section 2.2 covers different approaches within a full-text search by presenting the different techniques used to search documents.

Section 2.3 contains three different use case studies regarding the implementation of the ELK Stack, pattern detection within console logs and semantic search, and an implementation of a prototype that will improve effectiveness of searching documents.

2.1 The data mining process

Data mining [2] is the process of analyzing data from a dataset. Some information found in patterns may regard uncompleted purchases due to system failure. Different data formats are mined such as text, categorical and quantitative. Categorical data belongs to different categories such as children, teenager or parent. Quantitative data include numerical values such as height and age. The raw data is collected (preprocessing phase) and analyzed (analytical phase) in the data mining process. There are different forms of data mining called classification, clustering, association pattern mining and outlier detection.

2.1.1 The preprocessing phase

In the preprocessing phase [2], raw data is being structured. This phase prepares the data for the analytical phase. The preprocessing phase consists of three stages: feature extraction, data cleaning and feature selection and transformation.

1. Feature extraction: Analysis of data helps decide the most important parts that must be extracted. For instance, if it is a fraud detection application then the analyst should look at specific patterns that most likely indicates fraud.

2. Data cleaning: After collecting and extracting the data, some data may be missing or contain error. Therefore, in this stage data is being dropped or corrected.

3. Feature selection and transformation: In this stage methods are being used to correct very high-dimensional data such as image processing. This is necessary because those features may lower the efficiency of the algorithms used in the analytical phase.

2.1.2 The analytical phase

In the analytical phase [2], data is being analyzed, errors are being rectified and patterns explored. By analyzing data, algorithms that are best suited for detecting problems in systems can be found. This phase includes various techniques such as event
and host clustering, anomaly detection, root cause diagnosis, data extraction and dependency inference.

2.1.3 Association rule mining

Association rule mining, ARM, is used to find associations in data using algorithms such as Eclat and Apriori. Association rule [3] is a set of rules given by statements and conditions that will predict the occurrence of an item among several items. An example can be “If a log message contains 500 bytes, then it is 70% likely that it also will contain response code 500”. When an association rule is calculated the measurement of support, confidence and lift shows how reliable the association rule is.

The support [4] measurement is defined as,
\[ \text{Support}(X \rightarrow Y) = P(X \cup Y) \]

The confidence measurement is defined as,
\[ \text{Confidence}(X \rightarrow Y) = P(Y|X) \]

An example of data from which we can derive support and confidence values is shown in Table 2.1. In the example, Y is going to represent the item thesisWork.png and X is going to represent the item response code 500.

Table 2.1: Images together with their corresponding response code. Source author.

<table>
<thead>
<tr>
<th>Image</th>
<th>Response code</th>
</tr>
</thead>
<tbody>
<tr>
<td>thesisWork.png</td>
<td>500</td>
</tr>
<tr>
<td>dataMining.png</td>
<td>500</td>
</tr>
<tr>
<td>thesisWork.png</td>
<td>500</td>
</tr>
<tr>
<td>textMining.png</td>
<td>200</td>
</tr>
</tbody>
</table>

The support value counts how many times items appears together in a transaction, denoted as a row in the table. The union of X and Y indicates the probability of the transaction containing both thesisWork.png and response code 500. By looking at Table 2.1 the probability is 2/4.

Confidence is being measured by two different calculations. The first calculation measures how many times X appears in all transactions, which is 3/4. The second calculation measures in how many of those (the transactions where X appear) Y appear, which is 2/3. So, the probability that the transaction that contains X also contains Y is 2/3. The association thesisWork.png -> 500 becomes an association rule containing 2 items, the image and the response code.

Lift is a measurement which ranks the association rules by looking at the quality of the rule. Lift is used within a correlation analysis. In a correlation analysis, association rules are being evaluated by looking at the correlation between the items in
them. Since the correlation cannot be calculated by the support and confidence values, lift is used. By analyzing the correlation, misleading association rules can be removed. An example of an occurring misleading association rule can be if recorded transactions of the products Mac and PC among others, bought by costumers, are going to be analyzed with ARM. There are 5 000 transactions in total, whereof 3 950 transactions include PC, 4 650 transactions Mac and 2 900 includes both. Minimum support value is set to 40% and minimum confidence is set to 70%. The support value is 58% and the confidence value 79%. In general, Mac and PC computers are not correlated, that is, if a costumer walks into a store and buys a Mac there is not a high possibility that the same customer buys a PC. In fact, buying a Mac decreases the possibility of buying a PC. The probability of a customer buying a Mac is 93% (4650/5000=93%). By only looking at the confidence and support value for this association, it is easy to make wrong decisions. Since both values exceeds the minimum support and confidence thresholds the association rule is valid but it still may be misleading due to the correlation between the items. When calculating the lift value, three probability values are needed. The probability of a customer buying a PC (P(X) = 3 950 / 5 000 = 0.79), Mac (P(Y) = 0,93) and both (P (X U Y) = 0,58).

The lift measurement is defined as,
\[
\text{Lift (X, Y)} = \frac{\text{P (X U Y)}}{\text{P(X)P(Y)}}
\]

Inserting the values calculated in this example into the lift formula,
\[
\text{Lift (X, Y)} = 0,58 / (0.79 * 0.93) = 0.789.
\]

By looking at the calculation, the value of lift is 0.789. If the value is < 1, it indicates that there exists a negative correlation between the items. If the value is > 1, there exists a positive correlation and if it is == 1, there does not exist a correlation.

In this case, there exists a negative correlation. This means that there is a big chance that if one of these items occur the other will not because of the high individual probability of each item (Mac = 93%, PC = 79%).

2.1.3.1  Apriori algorithm

The Apriori algorithm [5] iterates through the database and sorts out data that occur frequently. The Apriori algorithm follows the rule that if a nonempty subset is a subset of a frequent itemset then it must also be frequent. To find frequent item sets, Apriori scans the database several times.

Here follows an example which illustrates how Apriori finds (k+1)-itemsets. For easier reference, the k in k-itemsets stands for how many items that an itemset includes. In the example below, 1-itemsets is referring to {ItemX} and 2-itemsets refers to {ItemX, ItemY}. Before running the algorithm, a value must be set as the minimum support count so that the transactions, shown in Table 2.2 below, that are not frequent can be sorted out. In this example, the minimum support count is 2.
In the first step a new table is being created where the datasets and their support count is being divided into two columns, as shown in Table 2.3. This is the first scan.

<table>
<thead>
<tr>
<th>C1</th>
<th>Support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Item1}</td>
<td>3</td>
</tr>
<tr>
<td>{Item2}</td>
<td>3</td>
</tr>
<tr>
<td>{Item3}</td>
<td>3</td>
</tr>
<tr>
<td>{Item4}</td>
<td>1</td>
</tr>
</tbody>
</table>

In the second step, the L1 – set of frequent 1-itemsets is being determined, as shown in Table 2.4. The itemsets that has a support count smaller than 2 is removed.

<table>
<thead>
<tr>
<th>L1</th>
<th>Support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Item1}</td>
<td>3</td>
</tr>
<tr>
<td>{Item2}</td>
<td>3</td>
</tr>
<tr>
<td>{Item3}</td>
<td>3</td>
</tr>
</tbody>
</table>

In the third step, the table C2 is determined. As shown below in Table 2.5, in each row there exist two items in a set and for each combination a new support count must be calculated. The order on how the items is being placed does not matter so {Item1, Item2} is equal to {Item2, Item1}. As shown in Table 2.2, Item1 and Item2 appear together in transaction 1 and 2, therefore their support count is 2. This is the second scan.

<table>
<thead>
<tr>
<th>C2</th>
<th>Support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Item1, Item2}</td>
<td>2</td>
</tr>
<tr>
<td>{Item1, Item3}</td>
<td>2</td>
</tr>
<tr>
<td>{Item2, Item3}</td>
<td>2</td>
</tr>
</tbody>
</table>

Now, in the fourth step, the same is done as in the second step but instead of finding frequent 1-itemsets, Apriori finds frequent 2-itemsets. The algorithm repeats step two and three until no more item sets can be found.
2.1.3.2 Eclat algorithm

Eclat algorithm [6] is like a depth first search, finding the elements by starting at the bottom and ending at the top. It is used to find associations between data in a set of transactions. The Eclat algorithm produce a frequent object once and by scanning the database only one time it becomes less time consuming. Eclat can only scan vertical databases, so if the data is in a horizontal format it transforms it into vertical, as shown below in Table 2.6 and Table 2.7. Eclat [7] uses the (k+1)-item sets as Apriori.

Table 2.6: Horizontal format. Source author.

<table>
<thead>
<tr>
<th>Transaction id</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Item1, Item3, Item2</td>
</tr>
<tr>
<td>2</td>
<td>Item1, Item2</td>
</tr>
<tr>
<td>3</td>
<td>Item1, Item3</td>
</tr>
<tr>
<td>4</td>
<td>Item2, Item3, Item4</td>
</tr>
</tbody>
</table>

Table 2.7: Vertical format. Source author.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Transaction id set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>Item2</td>
<td>{1, 2, 4}</td>
</tr>
<tr>
<td>Item3</td>
<td>{1, 3, 4}</td>
</tr>
<tr>
<td>Item4</td>
<td>{4}</td>
</tr>
</tbody>
</table>

Here follows an example on how the algorithm works. The minimum support count in this example is 2. The support count is measured by how many times an item occurs in different transaction ID’s, as shown in Table 2.2 and Table 2.3. Eclat removes the itemsets with support count less than the minimum support count, as shown in Table 2.8. It repeats this, adding k+1, until no more item sets can be found.

Table 2.8: Itemsets with transaction id set. Source author.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Transaction id set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>Item2</td>
<td>{1, 2, 4}</td>
</tr>
<tr>
<td>Item3</td>
<td>{1, 3, 4}</td>
</tr>
</tbody>
</table>

2.1.3.3 FP-growth

The FP-growth [8] is a depth-first algorithm that uses a data structure called FP-tree. In effortless way, there are two steps to the algorithm. In the first step a FP-tree is created and in the second step a search for items is performed. When building the FP-tree structure, transactions that have a support count lower than the assigned support are removed. Items are sorted in descending order based on support to put the most frequently occurring items close to the root. Each node also contains the support level of the frequent itemset. FP-growth reads transactions and maps them in a tree structure one at a time. The support level of each node is added by one as a new transaction passes through an already existing one.
To find the items in the tree structure the algorithm uses a header table. The header table contains all the distinct items with their support level and a pointer to find them in the tree structure. It shares much similarities with the Eclat algorithm but differs in how to calculate support level.

### Machine Learning

Machine learning [9] is about learning a machine to recognize unknown data through looking at distinctive characteristics. A machine can be learned to use different approaches. One approach is to use test data together with its corresponding solutions as input and learn the machine to recognize those relations so that, when the machine is trained and is going to execute its real work, it will know what to do with the incoming data.

#### Supervised Learning

Supervised Learning [10] is when learning a machine to detect and label data. An example could be assigning incoming mail with the labels spam or not spam. To learn a machine to know which label to put on an incoming mail, it must first be taught what the description is for one of the labels. For example, spam mail can have characteristics such as specific IP addresses and content (CLICK HERE, FREE!!). Those characteristics can then be used as a description for that label. Supervised learning includes two different problems, regression and classification.

#### Classification

Classification [10] is a process that contains classes, inputs and a classifier. The main goal in the process is to assign the right label to the right input data. This is the classifier’s task. K-nearest neighbor, CART and Random forest are examples on different classification algorithms.

#### Regression

Regression [10] is a method learning a machine to detect unknown data and label that data with a numeric value. For example, a machine that is going to predict the age of a human.

#### KNN

The K-nearest neighbor [11], KNN, is a pattern recognition algorithm. KNN is a simple machine learning algorithm that uses lazy learning, the learning is done when the query begins. The object uses the neighbors to decide classification by putting it to a vote, the weight of each vote is calculated by measuring the distance to the neighbor. For instance, if an object only has a single neighbor (k equals 1), the object in question are then assigned the same class as its only neighbor. The K in KNN is a user-defined variable that decides the number of neighbors that should be included in the vote.
2.1.4.1.4 CART

CART [12] is a decision-tree algorithm. A decision-tree algorithm uses different rules to decide the classification. It can be defined as a binary decision tree, it starts at the root node and answers a question assigned to the node. If the answer is correct it moves on to the left and if it is wrong it moves to the right, it continues until it reaches the top. The decision tree is defined during the learning process, it decides the question on its own by searching all the variables and then decides the question for the most optimal balance.

2.1.4.1.5 Random forest

The Random forest [13] algorithm is basically a combination of a decision-tree and KNN. Each tree in the forest is a unique decision-tree and the object that is about to be classified is going through all those trees. When that is done all the trees vote and the class with highest number of votes is assigned to the object. By having many different decision-trees a high number of different classes can be kept without being lost in the masses.

2.1.4.2 Unsupervised Learning

Unsupervised learning [9] is about training a machine to recognize unknown data by finding the characteristics of the data on its own. Within unsupervised learning, the most common method is clustering analysis. Clustering is when partition input data into clusters.

2.1.4.3 Machine learning tools

There are different tools [9] available that enables the possibility to learn and experiment with different machine learning algorithms. Some of those tools are: Python, R, Spark and Matlab. Spark is the only tool amongst them that is a distributed platform, which is needed when the dataset cannot fit in the computer’s main memory. It is also the only one that does not provide visualization. Python, R and Matlab supports only their own languages while Spark supports Scala, Java, Python and R. Python, R and Spark are open-source.

2.1.4.3.1 The R language

R [14] is a programming language that is often used for developing with high-quality analysis. It is used to implement data mining methods such as text mining, classification algorithms and association rule mining algorithms. R is the world widely used programming language for predictive analysis and statistical computing. Because R was created by and for statisticians it includes a variety of packages containing many statistical functions. R comes with its own packages at the first install, but there exists possibility to extend additional libraries in R. R uses data frames that has a natural data structure which is supported by few other programming languages. R can track unknown values in other applications which makes it easier to insert, save, reuse and share new analytical techniques between developers and data scientists. R enables the
use of advanced reading techniques and plotting of charts, lines and points, maps and 3D surfaces.

Whether R is used for optimizing genomic sequences, analyzing or predicting failure-times in a component there is a lot variety of different solutions published thus it is open source-code. The contributors of the latest version of R, are people located in different places over the world.

2.1.4.3.2 RStudio

RStudio [15] is an open-source software environment that is supported by many operating systems such as Windows, Mac OS X and numerous of UNIX platforms. RStudio uses its core scripting language but additional languages such as C, C++, Fortran and Java can be used in the code for execution or used as an exporting tool which are gained by other languages. RStudio has user-defined functions which can be extracted through a browser or desktop. In the desktop application, the forms are visualized through an html widget together with a cross-platform combined with a user interface framework.

2.1.5 Text mining, term frequency

The World Wide Web (WWW) [16] consists of a huge amount of text documents. It is difficult for a human to read every document in the web and to find similarities in them such as frequent topics. Here is where text mining is useful. The text documents that are being examined are first put together as one text and then they are going through a preprocess. The preprocess can contain stemming (removal of endings in words, if there are words in different forms such as common and commonly, then “ly” is being removed), removal of stop words (and, or), removal of characters (white space) and so on. In the preprocess, an analyst can manually remove strings that he or she considers irrelevant. A relevant string can be anything that helps the analyst tracking error. For instance, a specific image that has been requested. An irrelevant string can be a general word or string such as “time:” and “feb”.

2.2 Full-text search

Search engines such as Google [17] provides scalable solutions for efficient and flexible search of web documents. By ranking web documents the most valuable search result is presented. Searching for documents belonging to specialized domains using vocabulary and implicit background knowledge improves accuracy of the search result. The precision of the result could be improved using disambiguation and the searches could be improved using query variation that is meaningful.

Verbatim searches can be used in the English language grammar implicitly and in the documents. Eliminating affixes and stop words, using proximity based searches, a progress finding semantic invariance due to word inflections and permutations could improve searches. Synonyms improves searches but too many synonyms can neglect precision.
Latent Semantic Analysis (LSA) is a technique that regroups documents retrieved after most frequent occurrences of correlated words found in the document. LSA may retrieve documents which lacks some keywords but skip documents that matches keywords in a different context.

There exist diverse types of queries which are called keyword based syntactic queries and concept based semantic queries that take use of linguistic information (such as synonyms) explicitly and domain specific information (such as association and correlational word contained) in the document implicitly.

It is of importance to locate and highlight portions of text from the context retrieved by the query because the user may not be aware of the rest.

Information extraction is a very important topic to searching and indexing tools. Information extraction can be used, mapping phrases in documents to organizing subsequent results using knowledge from vocabularies (controlled vocabulary terms) using this step below.

(I) A controlled vocabulary is used when gathering terms that matches the user query phrase.
(II) Use these vocabulary terms to see which appears in the documents.
(III) Lastly collect and choose the most customized fractionally matching result term.

2.3 Related work

In this section, previous work is being presented regarding implementation of log management tool, the process of finding and analysing patterns in console logs and an approach to improve the effectiveness of searching for information within documents.

2.3.1 Implementation of the ELK Stack

“Samling, sökning och visualisering av loggfiler från testenheter” [18] is a thesis work about implementing the ELK Stack.

The implementation of the Elastic Stack was successful but there were some obstacles that had to be removed before everything could work. The format on the logs that the authors fed the Elastic Stack with was HTML and tar.gz which Logstash could not extract, so an additional script had to be implemented. The script had problems to parse individual lines from the HTML logs because of different structure. Another drawback with the script was that it slowed down the performance. Regarding Elasticsearch the authors pointed out that it is doing the measurements internally so it is not possible to know how precise Elasticsearch is when it is calculating. The result of the thesis work showed that an implementation of the Elastic Stack could be successful but that some obstacles could appear during the implementation process that had to be removed.
2.3.2 Finding and analysing patterns within console logs

“Online System Problem Detection by Mining Patterns of Console Logs” [19] is a research paper about finding and analysing patterns in console logs through data mining and statistical learning. The analysis of those patterns would detect potential abnormal execution traces. The solution in the paper differed from other similar solutions due to the implementation of a two-stage detection system. The two-stage online anomaly detection process captured patterns and identified problems. Before the implementation of the two-stage process, a pre-process which removed unnecessary data was done on the console logs.

A data pre-process that structured unstructured data had to finish to make the pattern recognition easier. In the pre-process the logs were parsed, relations between messages and program object were found, which created traces that were being converted to a numerical representation. An example of a trace could be a group of messages that described events such as opening and writing to the same file. When the pre-process was done, it was time for stage one in the two-stage process.

In stage one frequent pattern mining was implemented. The authors of this paper defined a frequent pattern as a subset of events that were closely related. Using their own algorithm which used frequent patterns on the results was helpful.

In stage two the authors implemented a Principal Component Analysis (PCA), detector on the non-pattern events from stage one to detect unusual behaviour patterns in the data.

The result showed that a two-stage data mining technique could be implemented and used to find common patterns from console logs that contained free text messages. A PCA detector could be used to find problems soon after they had occurred.

2.3.3 Improve effectiveness of searching for information within documents

In this section, a research from a paper is used which presents methods to improve searching inside documents for information. Integrating components such as Lucene APIs, LSA techniques, WORDNET and domain-specific controlled vocabulary searching for information became more effective. Finding document characteristics with LSI and incorporating language by Wordnet, precision of the search result was enhanced from the queries.

2.3.3.1 Architecture of prototype and implemented libraries

2.3.3.1.1 Java WordNet Library

Java WordNet Library (JWNL) [20] is an API used for retrieving verbs, adjectives, nouns and adverbs from a lexical database called Wordnet. Wordnet constructs words in the English language to a set of synonyms called synset, each representing a lexicalization.
Polysemous is a generic term for different senses of a word and exists in multiple synsets. Most synsets has a relation to other synsets when used as a verb, adverb, adjective or noun.

### 2.3.3.1.2 JAMA Library

JAMA [21] is a linear algebra package for construction and manipulation of dense matrices for Java. JAMA includes five fundamental matrix decompositions:

- Cholesky Decomposition of symmetric, positive definite matrices
- QR Decomposition of rectangular matrices
- Eigenvalue Decomposition of both symmetric and nonsymmetric square matrices
- LU Decomposition (Gaussian elimination) of rectangular matrices
- Singular Value Decomposition (SVD) of rectangular matrices

### 2.3.3.1.3 Configurer and DocumentIndexer

Latent Semantic Analysis (LSA) [22] is a mathematical method used to analyze relationships between documents through terms. It uses a SVD to construct a semantic space for frequently-used words. SVD calculates a term-document matrix using the JAMA library and results from the calculations are used within LSA. The rows within the term-document matrix is used to match terms and the columns is used to match documents.

### 2.3.3.1.4 Searcher and Query Modifier

A searcher object [17] is a query used to match indexed documents or the controlled vocabulary even called Domain Library (DL). Using Lucene’s own query representation (Api) the searcher object can translate all user queries. A Configurer specifies directories on which index a searcher object should run queries on, depending on the data source. A query modifier objects runs queries using Lucene’s own query in conjunction with searches using information such as synonyms from WordNet. If a search result matches a wide set of synonyms a Query Modifier can use heuristics to choose synonyms for query expansion (rearranging queries to improve recall). A query modifier objects uses proximity values (finding occurrences from one or some separately matching terms that are within a distance) for input with phrase query.

### 2.3.3.1.5 DL Term Locator

A domain library [17] (dl) is a domain specific controlled vocabulary that includes a set of domain library items. Every item contains a sequence of terms.

### 2.3.3.1.6 Match Highlighter

If the searcher query [17] finds hits from the documents, a Highlighter creates a file which stores the matched terms. The highlighter also presents excerpts from the
matched terms in the document hits. DL matches differs from the Highlighter due to the size of the DL-files. DL item is directly stored as matched terms in an output file created rather than reproducing the original DL-file with tagged matches. The DL Term Locator highlights occurrences of terms found and DL-items in the document.

2.3.3.2 An architecture of a Content-based indexing and Semantic Search Engine

The indexing and semantic search prototype [17] uses a document collection and sets parameters through the configuration files.

- Further steps in the process is to create and maintain indexes for the collected documents to improve searches with Lucene. The inverted index is used to store common terms from documents with LSA using the JAMA library.
- For the user queries and options, the prototype performs matches of phrases with help of morphological processing using a porter stemmer, matching of wildcard patterns, Boolean queries, expanding search terms using WordNet which is used through the JWNL library and LSA by WordNet via JWNL library, executing proximity queries, and uses LSA techniques to determine relevant documents.
- The search result should be highlighted with a relevant portion of the full text.

The prototype was developed to index and search domain-specific controlled vocabulary of terms. These terms are retrieved from xml files and used for semi-automatic content extraction which is a program that helps detect entities in the text, relations between entities such as a person and events mentioned in the text.

2.3.3.3 Query Effectiveness

Searches was made established syntactically and semantically in the Medline database:

(i) “Syntatic variations (e. g., stemming): “Test certificate” was words that query matched document phrase such as: “certificate of test, “test certification”, etc. likewise, “dia*” matches “dia”, “and diameter”, etc, “acc* level quality” matched “Acceptable level of quality”, etc.

(ii) Semantic invariance for (example using synonyms): The matches using queries with keyword “Tensile strength” matched document phrases “ductile force”, “part number” matched “lot number and part number”, “mold” matched “castings”, “cast”, “forging” and “forge”, etc. “causes cancer” matched “induces cancer”, “Insufficient immunity” matched “immune deficiency”, “reasons for cancer” etc.”

Under the comparison by the impacts of LSA, the prototype was tested by the authors using searches with the MEDLINE collection. Because of the resource limitations there existed over five thousand index terms from the five hundred documents inside the collection. A hundred-factor SVD was assigned on the matrix and stored for further searches.
2.3.3.4 Modularity through extension and reuse

The documents collected by the search result should be grouped after word senses to boost accuracy using group labels [17]. Aside that it was not easy to express or determining word senses, due firm similarity, the author explored that the grouping of documents by finding synonyms in the documents and assigning a member to the synonyms. To improve the capability of query entries and the organisation of query results the author decided to use two methods to their indexing and search tool: The query was processed by a spell checker called Jazzy using the Java framework. Even the search results containing documents was grouped in a hierarchy based on the occurrence of the synonyms. First, the input query was processed through Jazzy, a Java Open Source Spell-Checker. Second, the flat list of retrieved documents was grouped in a hierarchy based. This grouping was arranged due to the number of synonyms found in a hierarchy, this to enable deletion or leave certain subgroups of documents to improve relevance of found word senses. The Author searched for the keyword “deficiency”, resulting with documents containing the phrase “insufficiency”, “lack”, “deficiency” and “want” using the MEDLINE collection. The documents contained in the folder labelled “want” were skipped due to that the word "want" was out of context.
3 Methods

The present section consists of four different sections. Section 3.1 gives an overview of iStone’s system that generates logs, section 3.2 gives an overview over two search engines and their methods as well as a comparison. Section 3.3 encompasses an analysis of various log management and analysis tools and important factors such as techniques, performance and reliability. Section 3.4 contains the implementation of the chosen log management and analysis tool, the ELK Stack followed by section 3.5 which presents the implementation of the different stages in the data mining process.

3.1 Overview of the current system

Hybris Software [23] is a platform for e-commerce which iStone uses. Hybris started first as a PIM, Product Information Management, solution which later included the capabilities of e-commerce. Hybris offers PIM as a standalone solution but it is common among iStone’s customers to combine PIM together with e-commerce modules. Hybris uses combined and frequently integrated PIM and e-commerce that is built on the Java architecture and data model. iStone has Hybris Software as a partner and uses their services. The logs examined in this study were generated by Hybris.

![Figure 3.1: An overview of Hybris. Source iStone.](image)

3.2 Choosing the right search engine

Section 3.2.1 includes the techniques and methods provided by Apache Lucene which is a high-performance text search engine library. Section 3.2.2 contains an overview of the search engine Sphinx and finally, in section 3.2.3, a comparison between Lucene and Sphinx is presented.
3.2.1 Techniques used in Lucene

Lucene [24] is a search engine library which supports full text search. Searching in Lucene gives a result that is sorted in descending order per relevance. The relevance of each document is determined by a score value (always a float value) called _score. This _score is generated by one of several possible query clauses such as a fuzzy query. Fuzzy queries measure similarity of spelling terms in the documents to your search term. Another query clause called Term query calculates the occurrence of word found in the documents to your search term. This query is useful to gather statistics of the terms found. By relevance, the algorithm measures the similarity in contents between a full-text field and full text query string.

The Boolean model handles some of the Boolean operators such as "AND", "OR" that can be used in your query. A query words such as: "Elastic AND Stack OR Splunk" will retrieve documents that includes the terms Elastic Stack or Splunk.

By given occurrences a term appears in the field it gets ranked. If a term appears multiple times within a field than once the term is considered more relevant. The term frequency is calculated by the formula: tf(t in d) = √frequency. The t stands for term and d for document, these two frequencies (tf) where t stands for the occurrences of a term that is contained in the document. The quantity of t times d defines the term frequency.

An inverse document frequency(idf) is a technique that counts the occurrences of a certain term in documents. The more a term appear in various documents the lower weight it gets. Common words such as “and” and “or” is common in many text documents and therefore irrelevant when ranking. Words such as database or Java is more relevant and makes more relevance and is of interest when searching in documents.

The formula for idf is denoted: idf(t) = 1 + log (numDocs / (dsocFreq + 1)). The Inverse document frequency formula consists of the document frequency (idf) of t term. T is the logarithm for quantity of documents in the index, divided by the documents that includes the term.

A field-length norm is a technique used in full-text search to match terms in fields. A term found in a short field such as title field gives higher weight (relevance) that the content is about the term. The field length is used within the calculation and matched terms in a greater field length gives less relevance. The field length is added within several documents. The formula for field-length norm (norm) is denoted norm(d) = 1 / √numTerms [25], which is equal to an inverse square root of the number of terms in the field. The norm is an important value for full-text search and consumes 1 byte of each string and document in the index. Queries combined with TF/IDE score can be used with some factor to calculate some statistics. Such factors can be phrase or fuzzy-queries that offer term proximity, or term similarity-queries. Full text search engine must find documents and sort them by a score value.
METHODS

Using full-text formulas or similarity algorithms to search inside your document every document can be assigned its rank by score value. Boolean is a vital part when working with full-text search but it is enough alone, therefore a score value is needed to measure relevance. Different query clauses combined can act as a compound query like the Boolean query using different query statements. By inserting statements in a query, every statement must be true to retrieve a result.

When storing structured data such as string, numbers and dates in the database, searching becomes easy using queries that matches any document in the database. When measuring relevance, there is other principles used than full-text search that can be used; structured data is just as important to as input. For example, if an apartment is announced to be sold it should present some features. Features such as number of rooms, floor area, location, rent and house-price. These characteristics is important when searching for an apartment and makes the document more relevant.

Lucene use the Boolean model approach to find identical documents as well as a formula called the practical scoring function to calculate the relevance. Practical scoring function uses some of the approach from the term frequency/inverse document frequency and the vector space model to bring the features such as field length normalization, coordination factor, and term or query clause boosting. The vector space model provides various methods for comparing multiple terms to query documents. The returning result should give a score regarding how well the query matched the terms. A representation is a model where each query and document is a vector. The number in the vector represents the weight of the term.

3.2.2 Techniques used in Sphinx

Sphinx is a full text search engine [26] that will provide search functionality. It was developed to especially function good with storing data into SQL database through use of various scripting languages. To provide fast searches in Sphinx a special data structure is built to enhance queries. Sphinx uses two different indexing backends called disk indexed backend and RT (realtime) index backend depending on the task. Disk indexes provides maximum indexing and searching speed by keeping low use of resources as RAM. But there is a withdrawal, preventing updates of existing indexes and indexing of incrementally documents to a disc index. Just a batch of the whole disc can be done, rebuilding the disc index from scratch. RT indexes allows updating existing indexes and index documents incrementally to the full text index. Writing is fast, allowing indexes to be searched normally just after 1 ms. Every document that is indexed by Sphinx is assigned a unique id.

Sphinx supports queries such as Boolean syntax when searching for indexed documents, including operators such as “and”, “OR”, “NOT” and grouping is allowed. Implicit operators “AND” is automatically contained in a query like “log message” which really means “log AND message”.

You can choose to let Sphinx automatically simplify your Boolean expression such as
• excess brackets: (D | C | B)) is equal to (D | C | B)
• Common NOT: ((B !C) | (D !C)) is equal to ((B|D) !C)
• NOT AND COMMON: ((C !D) | (C !Y) | (C !K)) is equal to (C !(D Y K))

Like Lucene, Sphinx provides ranking (weighting) of each retrieved document for every given query. This ranking is important so that it can output the most relevant documents first in the page.

There is no single standard way to rank document, something that is relevant for maybe isn’t for user. Therefore, ranking is configurable in in Sphinx and uses a notion that is a so-called ranker.

A ranker measure the input for a query and produces a rank as the result. In layman’s term a ranker decided the appropriate algorithm used when assigning weights to the document.

3.2.3 Comparison of Lucene and Sphinx

Sphinx provides easy setup of installation for searching and indexing due to easy configuration. In a comparison of Lucene and Sphinx made by J. Kumar [27], a table containing 100,000 records was indexed to a Sphinx and a Lucene database. Lucene found 660 stop words and had indexing time around 2761 second using default configuration settings. There existed certain setting parameters such as mergefactor and maxmergedocs to be assigned which could improve indexing rate.

Sphinx did not find any stop words and had an indexing time that was 246 seconds using default configurations. In Sphinx, there is no need to use different id:s for collecting data since it gets a unique id for each document indexed. Compared to Lucene you must therefore assign a separate id for each document and force in uniqueness to a combined software. To make the searches, script was using certain word as input presenting an average time. The search in Lucene was done on two fields but through the whole index for Sphinx.

<table>
<thead>
<tr>
<th>Searches/Thread</th>
<th>Concurrency - no of simultaneous threads</th>
<th>Total searches</th>
<th>Total time (milliseconds)</th>
<th>Average time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUCENE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>673</td>
<td>134.6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>10</td>
<td>615</td>
<td>61.5</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>30</td>
<td>897</td>
<td>29.9</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>150</td>
<td>2762</td>
<td>18.41</td>
</tr>
</tbody>
</table>

If you reach a huge index such as 100,000,000 records with an index size of approximately 9 GB and with a concurrency of 100 searches at a time the author found that concurrency could be an issue. Single searches are therefore much faster in Sphinx than Lucene but we must consider that OR clauses was not used for the Sphinx
search. So, Sphinx didn’t need like Lucene to fulfill 2 different result sets and make a union of these.

Table 3.2 Result of searches per thread, number of simultaneous threads (concurrency), amount of randomly selected words used for search, total time and average time in milliseconds using Sphinx.

<table>
<thead>
<tr>
<th>Searches/Thread</th>
<th>Concurrency - no of simultaneous threads</th>
<th>Total searches</th>
<th>Total time (milliseconds)</th>
<th>Average time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>512</td>
<td>102.4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>10</td>
<td>733</td>
<td>73.3</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2272</td>
<td>75.73</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>150</td>
<td>4439</td>
<td>29.59</td>
</tr>
</tbody>
</table>

Pointed out Lucene improved rates with time and offers a lot of features. If you are need of storing a lot of data and performance is requested, then Lucene is the right choice. Sphinx has a good index time which remains small and free from problems. Even Lucene offers a lot of good features but Sphinx can be used if the data set is small and development needs to fasten up due to easy configurations and installation.

3.3 Choosing the right log management and analysis tool

Large systems often consist of several different components that generates data. Raw data can be unstructured which makes it hard to analyze and search in. When using data mining it is necessary to first preprocess the raw data. This is where a log management and analysis tool can help. Tools such as the ELK Stack and Splunk Enterprise collects the data, divides it into a common format, makes it searchable and visualizes it in a visualization tool.

The chosen log management and analysis tool implemented in this study was the ELK Stack. Below this choice is justified as well as a presentation of another choice on the market, Splunk Enterprise.

3.3.1 The ELK Stack

The ELK Stack [28] is a collection of open-source software tools built on Apache Lucene that provides key components within log analysis. The architecture of the ELK Stack is illustrated in Figure 3.2 below. The main tools, included in the ELK Stack, are:

- Logstash [29]: A data collection engine used to collect, parse and send log data to Elasticsearch.
- Elasticsearch [30]: A search engine that enables real-time deep searching and analysis of data.
- Kibana [31]: A visualization tool is utilized to visualize the data from Elasticsearch.
3.3.1.1 **Elasticsearch**

Elasticsearch [33] is a search engine that enables search and analyze of data in real-time and full-text search. The core concepts of Elasticsearch are:

- **Cluster**: A cluster consists of one or more nodes that contains the user’s data. Every cluster has a unique name which is used to identify the cluster when adding nodes to it.

- **Node**: Each node is a server that is identified by its name. The name is used when identifying the relation between servers in the user’s network and the nodes in the user’s cluster.

- **Index**: An index consists of searchable documents and just as clusters and nodes, it is identified by its name. The name is relevant when the user wants to modify or search in the documents included in the index. Operations such as search, update, delete and indexing are made by referring to the name of the index. Several indexes can be defined in a cluster.

- **Chards**: Chards are pieces of an index that Elasticsearch can be configured to divide the data into when the amount of data in an index exceeds the hardware limits of a node. The user can decide the number of chards that the index is going to be divided into.

- **Types**: Different types can be defined in an index. If the user wants to add documents that have fields in common into a category, that category is called a type.

Elasticsearch runs inside a JVM and because it is a demanding software it is important to put down time and effort on monitoring memory usage and the garbage collector. Garbage collector and JVM memory issues can affect the performance of search queries. Garbage collection times may increase if the heap size is set too high. In JVM memory management each Java process has a configurable limit on heap memory usage. Unused memory is freed with the help of a garbage collector.

When indexing a document, Elasticsearch uses a node with a primary shard. The
node first writes collected data to a transaction log called write-ahead log, then to segment files that are immutable through Lucene.

The web front end Elasticsearch-head can be used to browse and interact with Elasticsearch [34].

3.3.1.1.1 Lucene

Lucene [35] is a scalable and high-performance retrieval library developed within the Apache Foundation’s Jakarta project. Lucene enables full-text search and provides functions through its API. It offers customized structure of user data storage by providing functionality such as query and indexing.

The input and output structure is equal to tables, entries and fields in a database. By mapping to the storage structure of Lucene, a database can add the functionality such as indexing and search function that other databases lacks. Lucene is divided into two parts: indexing and storing of data, and presenting search results.

Lucene indexes the document object by building an IndexWriter object which stores and maintains an index then by determining the storage path and configuration parameters. Then a Document object is built, equal to an entry in a relational database and determines the object of each domain, equal to which column of the entries. Lucene has three domains for different data output requirements and domain-based attributes which is of importance.

(1) Indexable: Given an entry that is stored in inverted form.
(2) Analyzable: Indicates that each word inside the field is indexed as a term. If the field is not split the whole text is considered a term.
(3) Storable: Indicates that the content is stored inside the field in terms of words instead of storing it in inverted form.

Searching [36] inside large files that has not been indexed can be very challenging and by indexing the data, finding relevant information is simplified. Lucene maintains the index by building multiple indexes and merging them regularly. Lucene indexes each document in an index segment that is quickly merged with a larger one to reduce number of segments, fastening up searches. Lucene can merge segments into one which is effective for rarely updated index. To avoid conflicts, Lucene creates new segment instead of editing existing. When merging, Lucene writes a new segment and deletes the old one which helps scaling, speeding up the indexing and search capability, and gives good input/output for merging and searching.

Some databases can only search on single keywords but Lucene uses meaningful keywords such as "O’Learys near Kista" that is of relevance which is “O’Learys” and “Kista”. Lucene also allows language filtrating using the analyzer class that lets you target data in different languages. Some search engines support only indexing of text or HTML documents but Lucene supports a large variety of different file formats such as SGML, word processing – documents.
3.3.1.2 Logstash

Logstash [37] is a data pipeline that is used to process all kind of data through collecting and parsing. The input data is different log formats such as firewall logs, Windows event logs, and syslog. To convert these into one single format, various plugins are used. There are more than 200 plugins available, many of which are submitted by developers that have created their own plugins. Logstash enables geo mapping, pattern matching and dynamic lookup. The Grok language is used in Logstash when structuring the data, as will be explained in more detail below.

Grok [38] is used as a command line interpreter, script language, for queries and analysis by software that parses and extracts data. It has an interpreter which works as a relational calculator, especially in large factbases. Grok language is used to write programs based on Grok statements and those Grok programs are executed by the interpreter. Set theory, a branch that deals with formal sets as unit, contains logical functions from grok. Here is an example of how grok can be used:

```
4 games := { "World of Warcraft", "Counter-Strike" }
5 movies := { "Titanic", "The Lord of the Rings" }
```

Figure 3.3: An example of set constructors in Grok language.

As shown in Figure 3.3, these games and movies are two set variables which include different strings. These two sets are combined into a common set called entertainment, which contains the union of games and movies. Union sets are combined through intersection and subtraction. These statements are useful when using the Grok command line interpreter. When entering a value or expression such as games, the interpreter evaluates the line and prints out World of Warcraft and Counter-Strike. Just as sets, relations can be defined in Grok. Relations are sets of unordered pairs. For example, if a pair of persons (Jannica, Rickard) are siblings and a pair of two cats (KittyPurrPurr, Gossan) are siblings, then the relation can be the sibling relation between those two different pairs.

3.3.1.3 Kibana

Kibana [39] is an open-source visualization tool which visualizes data in a user-friendly interface where users can create graphs, dashboards, make searches and filter data. It enables the user to run and save custom queries within the time interval of their choice to find specific information. Boolean operators, field filtering and wildcards are used in the searches. The data is visualized in different forms of diagrams such as pie chart, data table and vertical bar. The unique features make it easier to understand large volumes of data.

3.3.2 Splunk

Splunk Enterprise [40] is a platform used for searching, analyzing and visualizing data. Splunk Enterprise is owned by the company Splunk that was founded in 2004 and is installed as a package. Splunk Enterprise [41] also comes as a SaaS, which is called Splunk Cloud. Splunk Enterprise and Splunk Cloud are both payment services.
Splunk Enterprise [42] offers a GUI (Graphical User Interface) and a query language SPL (Search Processing Language), used for searches. The results from the searches made by SPL are visualized through the GUI. Users can schedule saved queries and decide when they are going to be executed. Splunk Enterprise structures raw machine data generated by different technology systems such as servers, applications, sensors and networks. SPL is a combination between Unix pipeline syntax and SQL. Their usage includes analytical search, visualization and correlate data. SPL enables the possibility to use machine learning and anomaly detection.

Splunk Enterprise [43] operates through a pipeline where raw input data transforms into searchable events. The pipeline consists of 4 different stages: Input, Parsing, Indexing and Search. In the Input stage, raw input data is being divided into blocks where each block is assigned metadata keys such as host, source and source type. Splunk Enterprise also assigns the input data keys that will help to process the data in later stages. An example is the character encoding of the data stream. In the Parsing stage, the input data stream is being divided into individual events. By examining the data, Splunk Enterprise can identify or create timestamps, break the stream of data into individual lines and transform metadata and event data per regex transform rules. In this stage, the user can make own choices on how the input data should be indexed by customizing different actions such as applying metadata and masking sensitive data. In the Indexing stage, event data from the parsing stage is being divided into segments which makes it searchable. The processed data is stored in a flat file repository called index. In the Search stage, user actions such as how the user views, accesses and uses the indexed data are being managed.

In a large-scale deployment [44], Splunk Enterprise gets distributed and the indexer (Splunk Enterprise component) resides on its own machine. The indexer only handles the indexing of incoming data and searches. While in a small-scale deployment, the indexer also handles data input and search management functions.

3.3.3 A comparison of log management and analysis tools

Based on the outcome of the literature study a log management and analysis tool was chosen. Several log management and analysis tools were examined but in the end the choice stood between the ELK Stack [28] and Splunk Enterprise [43].

They are the leading enterprise solution approaches when it comes to log analytics. They are both well documented, they have reliable features and a big community that can help the user to set up and maintain the tool by being able to ask other users questions. Both solutions consist of a log parser, search engine and visualization software which are covering important areas regarding log analysis such as collecting log data, search capabilities, data visualizations. By providing a visualization software that is easy to understand and use, the ELK Stack and Splunk Enterprise opens for a broad user basis.

iStone has a large system that generates high-volume, unstructured and dynamic data. Therefore, it is important that the chosen log management tool to implement into iStone’s system is engineered to handle massive amounts of data. Elasticsearch, the search engine in the ELK Stack, can grow from a small cluster to a very large one without major complications. The larger cluster, the more planning is
being required from the developer. Splunk Enterprise scales to collect and index massive amounts of data every day.

Splunk, the company behind Splunk Enterprise, was founded in 2003 while Elasticsearch was released in 2010. By being in operation for a long time, both Splunk and Elastic, the company behind the ELK Stack, have had time to detect and solve bugs and flaws, upgrade software and features and add new features to the tools. By not having to spend time on building the core features of a log management and analytics tool, Splunk and Elastic can instead focus on being at the cutting-edge of log analytics technology. Both the ELK Stack and Splunk Enterprise are upgraded frequently with new versions.

Among other things, Google Trends was used to compare Splunk Enterprise versus the ELK Stack, as shown in Figure 3.4. The study in Google Trends show that the interest in searching for the ELK Stack has increased rapidly during a short period, from 2013 to present day unlike from the search for Splunk, which has increased slowly during a longer period, from 2006 to present day. It also shows that lately it has become more popular to search for Elasticsearch than to search for Splunk.

Splunk Enterprise is a commercial solution where the price is based on the volume of indexed data while the ELK Stack is open-source. Although the ELK Stack does not cost money to deploy it might not be entirely free. The company might have to set aside employees that needs to devote most or all their time on just maintaining the ELK Stack.

The ELK Stack is an open-source tool. By allowing developers to contribute, observe and test the code behind the products in the ELK Stack, increases the chances to spot flaws and correct them. By being open-source, the ELK Stack has been implemented by all types of companies, from small companies to big enterprises such as eBay. Until recently, Splunk has been targeting only big enterprises.

When installing the ELK Stack, the user must install three or four components, while installing Splunk Enterprise, the user only must install a package.

Relational databases [45] scales vertically and have rigid schemas where it is required to first declare the tables schema before inserting data into it. Non-relational databases [46], however, scales horizontally and have flexible schema design. They also provide high performance on big data sets and has no single point of failure which means that entire system does not shut down if one part of the system fails.

Horizontal scaling [47] is when scaling the system through adding more hardware or software while vertical scaling is when making an existing hardware or software more powerful such as CPU and RAM. Horizontal scaling distributes the load and reliability across multiple nodes which reduces the responsibilities of each node. This approach provides elasticity because if the load is increasing, new nodes are added to the system while the existing nodes are online. When using vertical scaling [48], the node must be taken offline when the load is increasing so that it can be adjusted to handle the new size of the load through an upgrade.

Since both the ELK Stack and Splunk Enterprise are built to scale and partly designed to handle large volumes of frequently incoming semi-structured and unstructured data, both solutions have chosen alternatives to relational databases.
Both Splunk Enterprise and the ELK Stack supports real-time search.

In short, Splunk Enterprise is a commercial self-developed system while the ELK Stack is built upon open-source components. Neither of the solutions use a relational database to store data, instead Splunk Enterprise use a data structure that is Splunk-built and the ELK Stack use a document-oriented database. Both solutions use horizontal scaling which is an important approach when developing a solution where scalability and elasticity are essential. Both have big communities but because the ELK Stack is open-source its community is bigger and highly active. Splunk Enterprise comes with much more features than the ELK Stack. Splunk is easier to install since it comes as a package unlike from the ELK Stack that is divided into three different components. Both solutions support real-time search.

Figure 3.4: A comparison of how frequently the search terms “logstash+elasticsearch+kibana” and “splunk” have been used in the Google search engine. Source Google Trends.

3.4 Implementation of Logstash

Logstash was used to collect, filtrate and send messages from the access - and console logs to Elasticsearch. iStone provided a zip folder containing logs generated over a period of five days. These log messages were divided into two different folders, one for access logs and one for console logs.
After implementing Logstash two configuration files were generated. The first file contained an input filter used for collecting, as shown in Figure 3.5, and filtrating the incoming logs. The second one contained the output which defines where to send the structured data. Logstash was configured so logs can easily upload to Elasticsearch through the terminal.

To prepare the log data for visualization in Kibana, two modifications were made in the input filter. The first modification was to convert the byte's field, describing the size of the data within the access logs into integers. The second one was to extract and divide the ping response times from the console logs into a scale.

The filter used to structure access log data, included a predefined grok pattern called COMBINEDAPACHELOG. COMBINEDAPACHELOG is a standard pattern for access logs generated by Apache HTTP servers. Since the grok pattern converted bytes into strings it was replaced by a customized pattern that converted the bytes into integers instead.

A scale was created through grok statements to get an overview of ping response times. Grok is used to extract and divide the different ping response times into a scale. The scale contained different values depending on how high the number was, as shown in table 3.3 below.

Table 3.3: Scale measurements. Source author.

<table>
<thead>
<tr>
<th>Ping number</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 10</td>
<td>0</td>
</tr>
<tr>
<td>10 - 20</td>
<td>1</td>
</tr>
<tr>
<td>20 - 30</td>
<td>2</td>
</tr>
</tbody>
</table>
Data mining is a process, containing three stages: collecting, preprocessing and analyzing. The two approaches that were used to cover the different stages were the ELK Stack and the use of R programming language in RStudio. The ELK Stack covered all three stages while the R programming language was used only for further analysis of the data. The R language was used to implement classification and association pattern mining.

3.5.1 The preprocessing phase

Collecting the data, that was going to be analyzed with data mining could be made by a software such as Logstash.

As was pointed out in section 3.4.1.1, the configuration file that was used for input data in Logstash contained grok patterns. These grok patterns divided the log messages into several fields such as response, bytes, request, verb, tags, geoip and clientip. After the filters were added, the structured log data was sent to Elasticsearch where it became stored and searchable.

3.5.1.1 Data Aggregation

Elasticsearch-head was used to get data from Elasticsearch clusters. Through the web, API, POST queries could be sent to the cluster that returned the desired log
messages, as shown below in Figure 3.7.

To be able to analyse the erroneous data, when implementing the data mining algorithms, the query narrowed down access log messages to messages that had generated HTTP responses: 500 Internal Server Error and 404 Not Found. As shown in figure 3.7, the query also narrowed down the fields to response, verb, request and country name. The returned dataset, containing log messages in JSON format, were opened and aggregated in RStudio. It was divided into different columns in a table, as shown in Table 3.4 below.

<table>
<thead>
<tr>
<th>Request</th>
<th>Response</th>
<th>Verb</th>
</tr>
</thead>
<tbody>
<tr>
<td>/favicon.ico</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/favicon.ico</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/zkau/web/zul/img/shadow-m.png</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/zkau/web/zul/img/shadow-tlr.png</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/zkau/web/zul/img/wnd/parol-corner.png</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/favicon.ico</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
<tr>
<td>/cockpit/images/stop_klein.jpg</td>
<td>500</td>
<td>GET</td>
</tr>
</tbody>
</table>

Table 3.4: Aggregated data - Log messages divided into columns in a table in RStudio. Source author.

### 3.5.2 The analytical phase

Kibana was used to get an overview of the log data. By using the search field in Kibana and a variety of different diagrams, it helped to decide which data fields to mine and which form of data mining to implement. Classification, association pattern mining and text mining were used to find associations, patterns and frequent words. The programming language R was used to implement the different algorithms due to
its variety of rich techniques. RStudio was chosen because it is a powerful IDE, open-source and a good program to plot in.

To decide what kind of information that could be considered valuable to know about in this thesis work, an analysis was performed. Since some employees at iStone had worked a long time with manual debugging, they had some knowledge about errors that could be generated within the system. Previous known cases that indicated error were: high ping numbers, response code 500 or 404 and high number of bytes within the response or request. A high number of bytes often indicated that large unscaled and uncached images was uploaded or requested. These indications of errors could be extracted and shown within the ELK Stack since they were known. They could be extracted within Logstash through dividing log data into data fields and they could be further divided (queries) and shown (diagrams, tables) within Kibana. There were also available plug-ins that could be used together with the ELK Stack to detect and notify everything that could be queried within Elasticsearch. Therefore, to learn a machine to only detect whenever a log message contained for instance response code 500 seemed to be an overkill. Instead of just looking at one individual data field, it is interesting to take advantage of the large range of features that machine learning provides and look at hidden patterns and relations between the data fields.

When analyzing the filtered log data, the focus was put on looking at the information in different data fields that had some form of relevant connection to errors. Since the debugging process aims to find and resolve errors generated by the system and since this thesis work focused on automating that process, using data connected to errors would be relevant to use as input parameters when implementing external algorithms.

The data fields that were chosen was: Response, Request and Verb. These fields contained objects, for instance images, requested by users, response code of the request such as 500 or 404 and if it was a GET or POST. The primary data field that was chosen was Response since it could contain the HTTP response code 500 (Internal Server Error) which implies that something has gone wrong. Alone, the response code does not give valuable information, therefore the other fields were chosen as well, so that the error message could be identified.

Association rule learning was used to find associations between the data fields. It was chosen since it could be used to find information from the data fields that could be interesting when debugging and contribute to an automatization of the debugging process.

A conclusion was made after analyzing the data extracted from Elasticsearch. It stated that it would be interesting to find a frequent pattern among an image, bytes and response code 500. The discovery of such associations would give the analyst information about frequently upcoming images that generates the response code 500. This is a hidden pattern that, by manually look at generated log messages, would be very difficult and time consuming to find. This insight would help analysts to prevent those images for continuing generate errors. Therefore, it is a suitable pattern to learn a machine to detect.
3.5.2.1 Cran

The Comprehensive R Archive Network, CRAN, is a network which stores documentation and codes which are used in R. The CRAN packages caret, tm, e1071, jsonlite, rpart and arules were used in this study. Tm is used for text mining. Two of Tm functions are Corpus, which collects text documents, and the interface tm_map, which removes punctuation, whitespace and so on [49]. Train is a method in caret whose functions include performance measurement and classification algorithms. Jsonlite was used in RStudio to read the JSON data from Elasticsearch. E1071, rpart and arules enables the possibility to run the different algorithms.

3.5.2.2 Association rule learning

When deciding which ARM algorithm to implement, the focus was put on highlighting the differences between the different algorithms. Since all ARM algorithms should generate the same output (patterns), the focus was put on the difference in their efficiency rather than the patterns they extract from the input data. A comparison was made between the three algorithms: Apriori, Eclat and FP-growth, due to their popularity found in literature.

There are different [50] things that affects the algorithm's performance. How algorithms perform their searches are one thing and another is the characteristics of the data set. Different characteristics of data sets such as the number of transactions and number of items within each transaction, plays a major role when it comes to the performance of different ARM algorithms. For easier reference, an explanation of data set characteristics together with examples from the data set chosen in this thesis work is presented in Table 3.5 below.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set</td>
<td>Input data</td>
</tr>
<tr>
<td>Transaction/Basket</td>
<td>A row in the table</td>
</tr>
<tr>
<td>Items</td>
<td>/favicon.ico, 500, GET</td>
</tr>
<tr>
<td>Frequent sets</td>
<td>Frequently occurring sets of items</td>
</tr>
<tr>
<td>Transaction/Basket size</td>
<td>Number of items per row</td>
</tr>
</tbody>
</table>

Since iStone’s system contains different components that frequently generates a lot of different logs as well as a large amount of log data, there is a high possibility that the volume of the input data will increase drastically. This was taken in consideration when choosing a suitable algorithm to implement.

An empirical study by Jeff Heaton [51] found that Eclat and FP-Growth both handle increases in maximum transaction size considerably better than the Apriori algorithm. In the study, a large data set was used as input data to the ARM algorithms Eclat, Apriori and FP-growth. The execution was presented in a diagram which showed the relation between the runtime of the algorithms and max transaction size of the data set.
Another study made by HooshSadat et al. [50] found that, by using a new technique called FARM-AP (developed by the authors), the fastest ARM algorithm can be predicted through a classifier. In the study the authors used FARM-AP to predict which of the different ARM algorithms: Eclat, Apriori or FP-Growth that performed the best on different data sets. FARM-AP predicted that both Eclat and FP-Growth performed much faster than Apriori in all instances of their simulation.

The studies presented above showed that both Eclat and FP-Growth outperformed Apriori significantly and that Eclat and FP-growth performed almost equally with a slight difference in runtime.

A survey on Frequent Pattern Mining by Goethals [8] found that the same ARM algorithms performed differently depending on the implementation. He stated that it was the cause behind articles coming to different conclusions about the performance of the same algorithms. In the survey, an in-depth analysis of many ARM algorithms was made. Through the analysis, it was concluded that a hybrid algorithm (combination between using Eclat and Apriori) was the most efficient algorithm when having a sparse database and that Eclat was the best algorithm when having a dense database.

Previous studies all addressed the fact that different characteristics of the datasets affects the performance of ARM algorithms and therefore plays a major role when deciding which one to implement.

Based on the surveys presented above, Eclat and FP-Growth were the ones performing best in general. But in the end, Eclat was chosen. The choice was based on the facts stated above that Eclat, not only was one of the best performing algorithms that effectively could handle large data sets, but that it also could be a part of a hybrid algorithm (a combination of Eclat and Apriori). By enabling two different approaches, contributed to a flexible solution adaptable for different scenarios. Therefore, this solution was suitable for this thesis work since data sets chosen from the logs generated by iStone’s system may vary.

Eclat scanned the input data to find associations between the items: response, request, bytes, and presented the results in a table containing five columns: lhs (left hand side), rhs (right hand side), support, confidence and lift. Each row contains an association rule. Eclat was implemented with R by the help of rules and the aggregated data, collected from Elasticsearch, was used as input. The result was modified so only the response code appeared in the rhs. Before running the algorithm, the threshold minimum support and max length was set. Minimum support, stated that only the association rules had larger support value than 0.015, would be presented as a part of the result. It also stated the association rules with the highest support values
should come first in the result.

```r
eclat_func <- function(Table1) {
  itemsets <- eclat(Table1,parameter = list(supp=0.015, maxlen=5))
  rules <- ruleInduction(itemsets, control = list(Verbose = FALSE))
  rules.sorted <- sort(rules,decreasing = TRUE, by="support")
  return(rules.sorted)
}
```

Figure 3.8: Eclat function written in R programming language.

3.5.2.3 Machine Learning

In this study, one of the objectives was to explore the possibility to use machine learning techniques to automate the debugging of log messages. Since Machine Learning is such a broad topic, The Algorithm Cheat Sheet from Microsoft Azure [52] was used to help narrow down the different approaches within Machine Learning.

![Overview over areas within machine learning](image)

Figure 3.9: Overview over areas within machine learning

There are different arrows coming out from the yellow START circle, shown in Figure 3.9 above, these arrows go to different approaches within Machine Learning. Beside each arrow there are overviews over the approaches. By having those overviews in mind, an analysis was performed to find an approach that would extract valuable information from the input data that could help to automate the debugging.

After finding relevant associations through ARM, a decision was made to explore the possibilities to combine those associations with machine learning. Since the associations gave valuable information about errors it seemed relevant to find an approach
that could make a machine detect them whenever they became generated within the logs. By having those associations together with the overviews over the different categories within machine learning in mind, an example of an approach was stated. The example was based on a combination between Supervised learning and the associations. Before a sample of the association rules could be used as an input dataset, it had to be labelled. To do so, the different association rules were assigned one of three labels: VALID, LESS VALID and NOT VALID. Association rules containing response code 500 or 404 together with high values on confidence, lift and support were assigned VALID. Association rules with the same response code but with low values on confidence and support were assigned LESS VALID. Association rules with other response codes such as 200 were assigned NOT VALID. The idea behind that setup was that a machine cannot learn to detect data through only one label. It must get in contact with other labels so that it can distinguish one from another.

The classification algorithms CART, KNN and Random forest was implemented with R. These algorithms were chosen since they all were algorithms used for classification. In Table 3.6 below shows the data used to test the algorithms.

Table 3.6: Table containing labeled test data used for supervised learning. Source author.

3.5.2.4 Text mining

When analyzing what kind of approaches that could be used to contribute to an automated debugging, text mining was investigated. Since insight in log data helps to find out why certain errors are generated, the text mining technique seemed relevant.

With text mining, frequent words can be found within the log data. If the technique were to be implemented on all log data generated by iStone's system, it would result in many frequent words. All of whom may not have a significant importance since log data contains so much information about things that goes right in the system. That would, not only, have an impact on the performance of the text mining algorithm due to the size of the input data, but also generate a large amount of unessential frequent
words that would drown out the words that could give valuable information about errors. Therefore, a decision was made that it would be relevant to narrow down the log messages and implement text mining on log messages that only indicated error. The input data chosen in this study was therefore log messages only containing response code 500.

Before implementing text mining, the log data had to be cleaned. Cleaning the data to find frequent strings reduced the time to produce relevant result. Specific strings, words and characters were removed and considered irrelevant. Strings such as “trying”, ”values:”, ”2016”, ”feb” and ”time:” together with stop words, function words such as “and”, were removed. The stop words were predefined but the strings specific for this study were defined manually. Other adjustments when cleaning the data were converting all characters into lowercase, removing punctuation and whitespaces. The tm_map function inside tm was used to clean the text.
4 Result

In this chapter, section 4.1 describes the result of the ELK Stack implementation, section 4.2 describes the associations found by Eclat, section 4.3 gives an example of an implementation of machine learning algorithms with the R language followed by section 4.4 which presents frequent terms generated by text mining.

4.1 The ELK Stack

The ELK Stack was successfully implemented. Logstash collected, structured and sent the data within the access - and console logs of two different index in Elasticsearch.

A configuration file, which defined the data fields in the logs, were added and modified in Logstash. A regular expression was included in the grok pattern file to extract the time it took to ping the JVM. Kibana enabled calculations on data fields in the logs.

Through predefined functions in Kibana, diagrams were generated by searches on the log data with specific time intervals as requested by iStone. After they had been generated they were presented on a dashboard, as shown in Figure 4.1 below.

As mentioned in section 3.4.1.1 Tasks made in Logstash, a scale was created through grok statements to get an overview of ping response times. The different ping times were divided into different intervals depending on how long it took to ping the JVM. The scale was visualized in Kibana. Most frequently occurring bytes, response codes and unique IP addresses per minute were visualized in circle diagrams. Most frequently occurring IP addresses from the access logs were visualized in a chart. This would help to detect suspicious activity in the network such as denial-of-service attacks from unauthorized IP addresses. The chart had different sections where each section contained an IP address. When clicking on a section, log messages related to that specific IP address were shown. The log messages contained the client’s interactions with iStone’s system.
Association rule learning

The association rules generated by the Eclat algorithm are shown in Figure 4.2. The input dataset contained log messages with either response code 500 or 404. All the rules had a confidence and lift value at 1. The lift value of 1 implied that all the rules were of excellent quality and the confidence value of 1 implied whenever items in the left-hand side appeared in a row in the input table, the items in the right-hand side would also appear. Support measures how often the items in left-hand side appears together with the items in the right-hand side in the input dataset. The support value at the first row was 0.9992 which implies the items 1091 bytes and response code 500 appeared together approximately 9/10 in the input dataset.

Supervised learning

The method train was used to explore the possibility to train classification algorithms together with measuring the performance. Train was used on the algorithms: Random forest, KNN and CART. Accuracy measures how close algorithms are to predict labelling of the input data. Under the accuracy headline in Figure 4.3 below there is a 5-number summary that includes the columns: minimum, 1st quartile, median, 3rd quartile and maximum. The median value measures the center of the data; the minimum and maximum values measures the range of the data and the 1st and 3rd quartiles are measurements on how the data is spread. For instance, if the input data was:
2, 5, 6, 9, 12. Then the 2 would be the minimum measurement, 3.5 would be the 1st quartile, 6 the median, 10.5 the 3rd quartile and 12 the maximum.

<table>
<thead>
<tr>
<th></th>
<th>Min. 1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>NA's</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART</td>
<td>0.5000</td>
<td>0.5</td>
<td>0.5478</td>
<td>0.7292</td>
<td>0.75</td>
<td>0</td>
</tr>
<tr>
<td>KNN</td>
<td>0.2708</td>
<td>0.5</td>
<td>0.4700</td>
<td>0.6667</td>
<td>1.00</td>
<td>0</td>
</tr>
<tr>
<td>rf</td>
<td>0.5000</td>
<td>0.5</td>
<td>0.5489</td>
<td>0.7500</td>
<td>1.00</td>
<td>0</td>
</tr>
</tbody>
</table>

Kappa

<table>
<thead>
<tr>
<th></th>
<th>Min. 1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>NA's</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART</td>
<td>-0.3333</td>
<td>0.1111</td>
<td>0.2756</td>
<td>0.575</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>KNN</td>
<td>-0.5000</td>
<td>0.0000</td>
<td>0.1930</td>
<td>0.500</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>rf</td>
<td>-0.5000</td>
<td>0.1333</td>
<td>0.3238</td>
<td>0.600</td>
<td>1.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.3: Performance measurement of the machine learning algorithms generated by the summary() function in RStudio. Presenting in-depth information about the performance of the different algorithms CART, KNN and Random forest. Source author.

All algorithms were implemented in RStudio and the CRAN libraries were used. CRAN library made it possible to execute each algorithm with one line of code as
shown below in Figure 4.5. The value in the method field was specific for each algorithm and became exchanged whenever a new algorithm was to be executed.

```r
fit.nb <- train(LEVEL~., data=TableJVM, method="nb", trControl=control)
```

Figure 4.5: Code written in R to train machine learning algorithms.

### 4.4 Text mining, term frequency

The tm package in R produced the most frequent words and how often they appeared in the log messages, as shown in Figure 4.6 and Figure 4.7 below. Using the word count function in RStudio a cloud of the most frequent words could be plotted, the frequent words that contained too many characters could not be plotted, instead they appeared in the list over the most frequent words.

![Figure 4.6: List over the most frequent words that appears in the log data. Source author.](image1)

![Figure 4.7: Wordcloud containing the most frequent words in the log data (except for very long words). Source author.](image2)
5 Analysis and discussion

There are several approaches when automating the debugging process, and since it is such a broad area that involves many parts of a system, it is not possible to automate the entire process. There were three approaches found suitable for automating parts of the debugging within iStone’s system: using a log management tool (The ELK Stack), ARM algorithms (combination between Eclat and a hybrid algorithm) and using machine learning algorithms.

There are a lot of factors left to be researched and evaluated. For instance, training the machine with a machine learning algorithm was left out, as well as the actual integration of the three approaches within iStone’s system. Only a prototype was made in this thesis work that was implemented and tested in a virtual machine. In theory, it is supposed to work, but when the ELK Stack is being installed within iStone’s system on several machines, it will most certainly cause new problems. Since the machines may have different specifications and different amounts of data.

5.1 The ELK Stack

The implementation of the ELK stack was successful. The fact that raw log data could be structured and searchable helped considerably when analyzing the data. The pre-process of the log data in Logstash and Elasticsearch facilitated the implementation of data mining algorithms, to the extent that it made the data searchable and allowed only the desired data fields when applying data mining algorithms.

By collecting common error sources such as most frequently occurring bytes, most frequently occurring client IP addresses, and ping response time and presenting them in a dashboard in Kibana, it gives the analyst an overview. The analyst can look at the dashboard instead of manually going through each error source to exclude the ones that are not relevant. So, by looking at the different diagrams in the dashboard, the time to find the error source for the specific case was reduced.

By visualizing the most frequently occurring bytes in Kibana, the analyst can more easily see if there are any unscaled and uncached images that are generating load. The most frequently occurring IP addresses shows if there are any requests made to the website from outside of the market. By visualizing the ping values in a diagram, it simplifies the root cause analysis, a method used to identify root causes, considerably. This means that iStone can match that information directly against running background jobs, for instance, and identify if one of those are the reason for increased ping response time. If iStone do not find a pattern, the next step is to look if the traffic on the website has increased.

Regarding the implementation of association and classification algorithms, input data should be adjusted after certain needs. The chosen input data in this study was used to demonstrate how algorithms could be used to automate the debugging. Before choosing the input data an analyst should analyse the logs. A visualization tool can help the analyst because it offers a good overview of the structured logs.
5.2 ARM and Eclat

When the integration was put on heavy load such as generating diagrams within Kibana and running queries to Elasticsearch the connection between the different components went down. It was required to collect a large amount of data from Elasticsearch since it was going to be used to find patterns within through ARM algorithms. Therefore, when generating large documents, it affected the virtual machine and the connection went down. Since the approaches presented in this thesis work was implemented as a prototype within a virtual machine, the data set generated was small in relation to what can become generated.

The associations shown in Figure 4.2 within section 4.2 were based on an input dataset only containing log messages with response code 500. If the dataset would instead contain all the log messages with response codes such as 200, 302, 500 and 404, the associations in Figure 4.2 would not appear. This is because there was a small number of messages with response code 500 in comparison to the number of messages with response code 200. In the column “confidence” in Figure 4.2 all the values are 1. The association algorithm first measured the number of log messages that the items in the left-hand side appeared in and after that the algorithm measured how many of that contained the response code 500. Therefore, in this case, the confidence value was always going to be 1. The right-hand side could be modified to show other items instead of only response code, so that the confidence values could differ. In this case, the focus was put on the support values, it was interesting to look at how many times the different items were associated with response code 500. Through this knowledge, patterns in the error log messages where discovered.

Within the ARM approach some preparation of the data set was required. After the data inside the JSON file, generated by Elasticsearch, was collected and put into a table inside RStudio, the data had to be prepared before it could be computed with Eclat. The preprocess was time consuming since it contained several steps of which there existed very little documentation. There existed examples of Eclat implementations with R but these examples used test data that already was preprocessed, so that part was left out. The R programming language was easy to use and simplified the implementation of algorithms due to the possibility to use different packages in CART. Data frames in R was one of the things within the preprocess that took some time to figure out. After putting the JSON file into a table in RStudio, the table had to be transformed into data frames and the columns containing strings (Request and Verb) had to be converted into factors. After figuring out which tasks that were necessary when preprocessing, the implementation of Eclat was made. The implementation was very easy as well as plotting the result.

5.3 Machine Learning algorithms

The result regarding supervised learning showed that machine learning algorithms are used to teach the machine to detect frequently appearing associations. The accuracy values did not differ a lot between the different algorithms that were compared. When training the classifiers to predict which label to put on the incoming associations, the input data must include a variety of different associations belonging to dif-
ferent labels. It does not work to have only a few examples because then the algorithms cannot learn. In this case, the input data did not contain enough examples but after adding more associations it enabled the possibility to train the algorithms. When implementing classification algorithms with the purpose of learning a program to detect errors, it would be beneficial to first analyse how and why previous errors occurred. Unfortunately, due to the limited time frame given for this study, and the restricted knowledge about why and how error appears in iStone’s system, there are no possibilities to train the classification algorithms. Further studies about how to train them to detect errors in the system, are therefore suggested.

5.4 Text Mining

Text mining was also used as an approach to help automating the debugging. The problem with using it on log data is that log data contains such a variety of frequently upcoming words that are irrelevant. Irrelevant defines, in this case, words which does not give valuable information by themselves, that must be put in a context to become valuable. Therefore, if this approach is going to be used, a pre-process must be made where those words are being removed from the text. There already exists code in the R language that can be used to remove common irrelevant stop words, as stated in section 3.5.2.4. Frequently upcoming words that are specific for iStone’s system must be defined by iStone. There may exist lists on the internet that contains common irrelevant words within web logs, which can be used as a part of the pre-process, but that has not been examined in this thesis.

5.5 Social, economic, environmental and ethical aspects

The results show that the use of a log management and analysis tool combined with a variety of data mining techniques can lead automating debugging. By using a visualization platform such as Kibana it can contribute to a better user experience.

When automating the debugging process, it decreases the time it takes to debug which can lead to fewer resources in terms of staff required for the new system. This decrease can lower the cost of resources used to troubleshoot. By looking at it from an environmental perspective, if it took several staffs to simultaneously debug when the process was manually done, it may reduce the number of computers intended for the automated debug.

The collected logs from iStone partly contained confidential information about their clients. Therefore, the authors avoid publishing sensitive information in this paper.
6 Conclusion

The main purpose of this thesis is to reduce the time it takes to debug a system through automating the process. This study indicates a combination between the ELK Stack and RStudio which can contribute to getting valuable insights in logs and automating the debugging of a system. Error sources can be efficiently identified using a dashboard in Kibana.

The exclusion of the error sources was manually done by iStone, but following the course of this research, that action will now be automated. This automation will save iStone about 1 - 3 hours of debugging depending on which information is found manually. With three debugging errands per week, this will save about 3 - 9 hours of work for iStone which is a tremendous upgrade and sets the premise for what can be established soon.

The ELK Stack structures the data, making it searchable and presents data, while association and classification algorithms enable further analysis. Through being able to find associations in the log data and teaching the machine to detect errors all by itself, it will give the analyst more insight into the data and it may reduce the time it takes to detect errors even more. More insight may lead to an understanding on why certain errors appears and how to prevent those errors from appearing in the future.

6.1 Future research

A future research project could investigate if the connection between RStudio and Elasticsearch can be automated. A hypothesis could be to collect the JSON data provided by Elasticsearch and use it in RStudio through a connection which will automate the process.
7 Bibliography

Retrieved 2016-04-23.

Retrieved 2016-05-05.

Retrieved 2016-04-09.


http://www.cs.ecu.edu/~dingq/CSCI6905/readings/zaki00scalable.pdf

[8] B. Goethals, "Survey on Frequent Pattern Mining",
http://adrem.ua.ac.be/~goethals/software/survey.pdf
Retrieved 2017-01-10.

Retrieved 2017-01-10.
[10] E. Alpaydin,  
"Introduction to Machine Learning", 2010,  

[11] L. E. Peterson,  
"K-nearest neighbor",  
http://scholarpedia.org/article/K-nearest_neighborLeif  

[12] R. Timofeev,  
“Classification and Regression Trees (CART) Theory and Applications”,  
http://edoc.hu-berlin.de/master/timofeev-roman-2004-12-20/PDF/timofeev.pdf,  

[13] L. Breiman, A. Cutler,  
“Random Forests”,  
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.html,  

[14] R. Sheldon,  
“Will the R language benefit from Microsoft acquisition?”,  
http://searchsqlserver.techtarget.com/opinion/Will-the-R-language-benefit-from-Microsoft-acquisition,  

https://www.safaribooksonline.com/library/view/getting-started-with/9781449314798/ch01.html,  


http://cecs.wrigvht.edu/~medal/people/vikram/files/wtas05.pdf  
Retrieved 2016-12-26.

[18] F. Rosenqvist, T. Henriksson,  
“Samling, sökning och visualisering av loggfiler från testenheter”,  
http://www.diva-portal.se/smash/get/diva2:819705/FULLTEXT01.pdf,  
Published 2015-06-10, Retrieved 2016-05-01.

[19] W. Xu, L. Huang, A. Fox, D. Patterson, M. Jordan,  
“Online System Problem Detection by Mining Patterns of Console Logs"
Retrieved 2016-05-01.

http://jwordnet.sourceforge.net/handbook.html
Retrieved 2017-01-28

http://math.nist.gov/javanumerics/jama/

http://www.siam.org/meetings/sdm06/workproceed/Text%20Mining/hendrickson22.pdf
Retrieved 2017-01-27.

[23] P. Sheldon, M. Goetz,
"The Forrester WaveTM: Product Information Management (PIM)",

[24] What is Relevance?

[25] Theory Behind Relevance Scoring
https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html,
Retrieved 2017-01-23.

[26] Chapter 5. Searching,
http://sphinxsearch.com/docs/current.html#searching
Retrieved 2017-01-23.

[27] J. Kumar, “Benchmarking results of mysql, lucene and sphinx"
http://jayant7k.blogspot.se/2006/06/benchmarking-results-of-mysql-lucene.html,

[28] “The Elastic Stack”,
https://www.elastic.co/products,
Published 2015-10-29, Retrieved 2016-04-29.

[29] "Logstash Introduction",
https://www.elastic.co/guide/en/logstash/current/introduction.html,
Retrieved 2016-08-30.
https://www.elastic.co/products/elasticsearch,
Retrieved 2016-08-02.

[31] Kibana, "Explore & Visualize Your Data",
https://www.elastic.co/products/kibana,
Retrieved 2016-08-02.

[32] M. Anicas,
“How to Use Kibana Dashboards and Visualizations",
https://www.digitalocean.com/community/tutorials/how-to-use-kibana-dashboards-and-
visualizations,
Published 2015-03-12, Retrieved 2016-04-29.

[33] Elastic,” Basic Concepts”,
https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html# near
realtime_nrt,
Retrieved 2016-08-29.

[34] Ben Birch,
“elasticsearch-head”,
https://mobz.github.io/elasticsearch-head/,
Retrieved 2016-08-29.

[35] Min Chang, Yuansheng Lou, Lei Qiu,
“An approach for time series similarity search based on Lucene”
Published 2016-12-19, Retrieved 2016-12-26.

[36] Brian Goetz,
“The Lucene search engine: Powerful, flexible, and free”
http://www.javaworld.com/article/2076176/java-app-dev/the-lucene-search-engine--powerful-
-flexible--and-free.html

[37] J. Turnbull,
“The Logstash Book log management made easy”,
https://www.logstashbook.com/TheLogstashBook_sample.pdf,
Published 2016-05-06, Retrieved 2016-04-29.

[38] R. Holt,
“Introduction to the Grok Language”,5, page 1,
http://plg.uwaterloo.ca/~holt/papers/grok-intro.doc,
Published 2002-05, Retrieved 2016-05-01.
[39] Rashid Khan, "The time series composer for Kibana",
https://www.elastic.co/products/kibana,
Published 2015-11-12, Retrieved 2016-08-02.

[40] Erik Swan and Rob Das, Splunk founders,
http://www.splunk.com/view/SP-CAAAGBY,
Published 2011-06-11, Retrieved 2016-04-29.

[41] Splunk Pricing,
https://www.splunk.com/en_us/products/pricing.html#tabs/ent,
Retrieved 2016-04-29.

[42] "Splunk Enterprise and Splunk Cloud",
http://www.splunk.com/en_us/products/splunk-enterprise/features.html,
Retrieved 2016-09-02.

[43] “How data moves through Splunk deployments: The data pipeline”,
http://docs.splunk.com/Documentation/Splunk/6.5.1/Deploy/Datapipeline
Retrieved 2017-01-10.

[44] “Scale your deployment with Splunk Enterprise components”,
http://docs.splunk.com/Documentation/Splunk/6.5.1/Deploy/Distributedoverview
Retrieved 2017-01-10.

[45] J. Bhogal, I. Choksi,
"Handling Big Data using NoSQL",
http://ieeexplore.ieee.org.focus.lib.kth.se/stamp/stamp.jsp?arnumber=7096207
Published 2015-04-30, Retrieved 2017-01-10.

[46] A. Dziedzic, J. Mulawka, "Analysis and comparison of NoSQL databases with an intro-
duction to consistent references in Big Data storage systems",
http://proceedings.spiedigitallibrary.org.focus.lib.kth.se/proceeding.aspx?articleid=1984639
Retrieved 2017-01-10.

[47] D. Beaumont,
"How to explain vertical and horizontal scaling in the cloud",
https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-
cloud/
Published 2014-09, Retrieved 2017-01-10.

[48] "Horizontal Vs. Vertical Scaling: Which Is Right For Your App?”,
https://www.g2techgroup.com/horizontal-vs-vertical-scaling-which-is-right-for-your-app/
Published 2016-02-22, Retrieved 2017-01-10.


