CHALMERS
UNIVERSITY OF TECHNOLOGY

# Log-Based Anomaly Detection for System Surveillance

Master's thesis in Computer Systems and Networks

CECILIA GEIJER
JOSEFINA ANDREASSON

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

# Log-Based Anomaly Detection for System Surveillance

CECILIA GEIJER
JOSEFINA ANDREASSON

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Log-Based Anomaly Detection for System Surveillance
CECILIA GEIJER
JOSEFINA ANDREASSON

Examiner: GRAHAM KEMP
Supervisor: DAG WEDELIN

Department of Computer Science & Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Gothenburg, Sweden 2015

## Abstract

As log files increase in size, it becomes increasingly difficult to manually detect errors within them. There is a need for automated tools for anomaly detection that do not require human assistance. This thesis aims to develop a prototype for such a tool that can be used to monitor the system state based on the produced log files. A specific and a generic approach for analyzing the data is explored to form a foundation for design decisions. Insights from the approaches are then used to build the prototype, which is done in three stages consisting of a basic prototype, extension of the prototype, and evaluation. The prototype is evaluated based on a number of interviews as well as through finding its accuracy and performance.

The resulting prototype graphs total lines, words and bigrams per hour. It visualizes the words, bigrams and anomalous messages that occur in each log file. A user specified blacklist highlights undesired words in any file. Anomaly detection is done by comparing historical and current values while taking the overall trends into account. The prototype was found to be useful by two professionals whose work involve log handling, and the interface was thought to be functional. It is able to correctly handle most data but suffers from false alarms, and found 11 out of 14 known errors. A shift in normality is handled well, and the prototype adapts within a week.

In conclusion, the developed prototype is usable, mainly for large log files. It requires more accurate anomaly detection, and the interface can be further improved.

**Keywords:** log analysis, anomaly detection, information visualization

## Acknowledgements

We would like to thank our supervisor Dag Wedelin for his enthusiasm and for helping us focus our ideas. We also give a special thank you to Rolf Snedsbøl who helped us through the ordeal of finding an examiner, and Graham Kemp for agreeing to examine us. Lastly, we would like to give our thanks to 3bits Consulting AB for their support in the development and testing of this thesis, especially to our advisor Jonas Hörnstein.

# Contents

# List of Figures

# List of Tables

# List of Listings

# 1

# Introduction

For a medium-sized e-commerce business, a single web server can easily produce up to a gigabyte of log files each day, far more than a human is able to sift through effectively. According to Gantz and Reinsel (2011), the amount of digital data produced will double in size every two years, quickly becoming a big drain on resources. There are many files to view and even more statistics to comprehend, especially when data is produced by a number of different servers, tools and systems. In the presence of errors, it becomes even more important for effective log analysis to be available in order to rapidly find the problem. According to Jayathilake (2012), the usual practice for investigating log files is to manually analyze them. It is thus desirable to revise how logs are presented to reduce the amount of information and time needed to maintain the system.

Logs could be analyzed in a real-time fashion to discover unusual errors before they cause failure or before too much damage is done. One example of this is inactivity alerts, another is actions that sometimes are perfectly legitimate but in other cases represent an error. Finding such problems as early as possible is vital to be able to deliver a good experience to the user.

There is a lack in tools not reliant on the user to make predictions about what normal behavior is, and that are suited for log files produced by a system rather than for example web traffic. There is a need for automated monitoring, something that emphasizes strange and erroneous behavior that is unlikely to be spotted by a human.

## 1.1 Goals

This master thesis aims to develop a prototype for automatic detection of errors and suspicious behavior through log analysis. It should discover and summarize anomalous points in a system and provide a quick overview of current system performance to the user. Focus is placed on automatic detection of errors and suspect behavior.

Though there has been a fair amount of work regarding how to handle log files, as seen in Section 2.5, this project places a lot of focus on the prototype working autonomously without human input, as well as considering the contents of the log files rather than any labels they have received. Tools that need human input require the user to know exactly what to look for in order to find all errors. This means that all possible anomalies must already be known to the user, and the tool would only be used to facilitate the search for those anomalies. In contrast, an autonomous tool would allow the user to find anomalies that are not previously known. This means that the user does not necessarily need to be an expert user of the system in order to use the tool.

One issue concerning anomaly detection of log files is how to classify behavior as normal. User behavior may change depending on the time of day, the date, or any number of reasons. The definition of normal thus fluctuates, and it is important that the system can take that into account. The first goal of this thesis is that the model of the system should be able to constantly update its definition of normality while still detecting anomalies.

Another problem is how to transform log files into a format that can be analyzed and manipulated. The logs are assumed to be stored in a text format, which must be mined for the information contained, for example by performing text classification. In order to minimize the amount of space used by the prototype, the information contained in the log files must be reduced. Therefore, decisions have to be made about which factors of the state are important to keep, and which can be discarded. The second goal is that log parsing should be accurate, whilst providing usable data for the analysis.

Accurate anomaly detection is an issue that is very important for this thesis. It concerns finding suspect behavior without relying on static boundaries or human interaction, which in turn relies on the definition of normality previously mentioned. The third, and most important goal of this thesis is thus to be able to perform accurate anomaly detection on the data from the log files.

One additional problem is how to present the information about the system to the users. Due to the large number of log files, this is related to the visualization of big data. The fourth goal is to enable the user to get a quick overview and easily spot anomalies. To do this, the amount of data presented needs to be reduced without removing any critical details that may be needed to draw conclusions.

A final goal is to make it possible to use the prototype on different systems. This is done by making as few assumptions as possible about the system, enabling use on systems having different structures and log files. As Ghoting et al. (2008) states, this is a balancing act between making enough assumptions to draw useful conclusions, and being general enough for files of different structures to be analyzed without the user having to adjust any settings.

The performance and usability of the prototype is tested using data consisting of logs from an e-commerce system provided by 3bits Consulting AB, as well as log files from a Windows 8.1 machine and a Ubuntu 14.04 machine owned by the authors. The prototype is evaluated based on the following questions:

- Is the prototype usable? This is a central question for the thesis, and involves the prototype being able to assist in handling problems that were previously more difficult to handle, and in what way it is an improvement. Missing functionality of the prototype should be considered here.

- What part of anomalies are detected? Points of interest are which anomalies that are not detected and why, as well as what false positives are present.
- Is the prototype able to take a shift of normality into account?
- How efficient is the prototype in terms of storage space used? This should for example consider how much space the prototype needs in comparison to the size of the original logs.
- Is the user interface comprehensible? This includes points such as whether the user is able to spot abnormalities in the visualization, and if the interface feels intuitive.
- Is the prototype flexible enough to be used on log files from different sources?

## 1.2 Delimitations

Data such as web traffic patterns is not used to determine the system state. This is because such data does not reflect the entire system, only parts available from the web. Information such as memory usage or processor load is not included either, as there are well established tools for such monitoring.

## 1.3 Structure of report

This report is structured in the following manner. Chapter 2 contains a walkthrough of the literature concerning relevant topics such as anomaly detection and visualization. It is followed by Chapter 3 which discusses the methodology used during the development of this thesis. Chapter 4 and Chapter 5 motivates design and structural decisions for two phases of the project. Chapter 6 describes the resulting prototype and its behavior and benchmarks, and is then followed by Chapter 7 which discusses the results while keeping the original goals in mind. Finally, Chapter 8 summarizes and concludes this thesis.

# 2

# Literature review

This chapter covers existing literature regarding some of the core topics of this thesis. Areas that are described are data mining, big data, anomaly detection and information visualization, as well as a brief summary of some existing log management tools.

## 2.1 Data mining

Data mining is the act of discovering new information from existing data. This can be done in a number of ways, and it is important to ask the right questions in order to gain information that is interesting and usable (Hay et al., 2008). In general, there are a few attributes that are desirable for a data mining algorithm, such that it should be able to handle different types of data or that it should be possible to normalize the data. Such an algorithm should also be efficient, and it should be scalable. It should be possible to present the results well, for example in a graph. If interactive discovery is desired it should also be possible for a user to adjust how the mining is done (Chen et al., 1996).

Some challenges related to data mining, especially log mining, are that there may be too little or too much data available, that files may be of different formats, and that files may be duplicated. Another challenge is how to know when something is interesting. One way is to remove expected events, so that only the unexpected remain. Depending on the data, interesting could be defined as rare events, top or bottom events, strange combinations, or strange counts of uninteresting things (Hay et al., 2008).

The first step to mining data is to define the goal; what questions are interesting to answer and how the resulting information can be used. Thereafter, data should be collected, preprocessed and cleaned, possibly reduced or transformed. Then, a suitable method should be chosen based on the available data and the goal (see Section 2.1.1). Once it has been applied, the results should be interpreted and possibly presented (Hay et al., 2008). It is important to note that any reduction in the data should happen after the data is collected, not before or during as this may lead to prejudice preventing new discoveries (Chuvakin, 2013).

### 2.1.1 Data mining methods

It is possible to mine data using somewhat uncomplicated methods such as measuring frequency, mean, standard deviation or standard error. It can also be interesting to look at what distribution fits the data, if any. Usually, it is necessary to establish a baseline or thresholds for the data, based on some weeks of past data. When this is done, it is important to age out old data after a suitable amount of time (Chuvakin, 2013).

Another rudimentary technique that can be applied to the data is string matching or regular expressions (Hay et al., 2008). This can be extended using pattern-based similarity measures to measure how similar two strings are (Chen et al., 1996).

Strings can also be divided into *tokens*, which can in its simplest form be seen as a sequence of characters split by a white space, roughly corresponding to a word. However, a good tokenizer also considers for example punctuation and whether it is part of a word or not. Splitting a text into tokens will lead to an increased understanding of its composition, by for example monitoring the frequency of the tokens (The Stanford Natural Language Processing Group, 2015; Jackson and Moulinier, 2007).

One aspect of data mining is whether it is done in real time or on historical data. Esmaeili and Almadan (2011) mention data mining of historical data logs as an alternative that requires the model to be updated continuously. Another alternative is to build the model using current logs, which they describe as stream data mining. Their paper is targeted towards network security, but it shows that the method of gaining information about the system is important to consider, especially when the behavior of the system may change over time.

#### Descriptive and predictive mining

Data mining can be divided into two different types: descriptive and predictive mining. Descriptive mining is focused on describing the domain, while predictive mining aims to make predictions about it based on the data.

One example of a descriptive approach is clustering, and another is frequent item sets discovery. Both describe what objects occur in different groups. Sequential pattern discovery is a similar approach, and states what objects occur in a specific sequence (Hay et al., 2008). A final example is association rule discovery, or how different parameters are related. Such a rule is only interesting if it is statistically independent and supported by the data, and Apriori algorithms could be used to do this rule discovering (Chen et al., 1996).

Instances of predictive mining are data classification, regression and deviation detection, which partly overlap with anomaly detection as described in Section 2.3 (Hay et al., 2008). Data classification involves labeling all data, and can be done based on decision trees. Unfortunately, the act of classifying data often suffers from scaling problems (Chen et al., 1996).

## 2.2 Big data

Though there is no real consensus about what exactly big data is, most would consider a bulk of log files to be big data. This is due to the large amounts of information autonomously and routinely produced and placed into logs. Jacobs (2009) suggests that one possible definition of big data is “data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time”, which stresses that as data handling capabilities increase, so will the amount of data. According to a survey by Schroeck et al. (2012), most consider one terabyte of data to be enough to be called big data today, and there is a need for methods that can handle the volume and variety of such data, preferably in real time.

Some example sources of big data are log files such as web requests or retail transactions, consisting of observations collected over a period of time (Jacobs, 2009). This is machine generated data that is collected but not analyzed, due to the generated data being too large to handle using traditional methods.

Big data often lacks advanced analysis, something that is needed to be able to gain insight from the data. In the survey by Schroeck et al. (2012), almost three out of four of those who worked with big data were analyzing, among other things, data from log files. This indicates a need for log file analysis, though it is far from the only relevant area. Of those who handled big data, 75% used basic methods such as querying and reporting, and 67% used predictive modeling. The emergence of big data has also put the need for more advanced data visualization into focus, both in terms of efficiency and the conveyance of complex information (Schroeck et al., 2012).

## 2.3 Anomaly detection

Anomaly detection is the task of finding data points that deviate from the rest of the data. It can also be called outlier detection or deviation detection (Hodge and Austin, 2004), and was according to Hill and Minsker (2010) historically done by hand by expert users of the system. Presently, a number of various automatic methods exist and can be used instead. Anomaly detection can be applied to a variety of different problems, such as using it for an intrusion detection system, for finding instances of fraud (Bolton and Hand, 2001), or for finding out if a safety critical system is running in an abnormal way before any major harm is done (Hodge and Austin, 2004).

Hill and Minsker (2010) stress the necessity of not using abnormal data to make predictions, as it would cause the predictions to in part view abnormal events as normal. It is also important to keep in mind how the alarms raised by the anomaly detection are to be handled. As Yamanishi and Maruyama (2005) mention, it is often difficult to spot a false alarm, and in a live environment it is probable that a human must check the system after every raised alarm regardless of whether it is a false positive or not.

There are a number of different methods that can be used to perform anomaly detection, and most can be grouped into being statistical, supervised, or unsupervised. Different methods require different types of data and are suited for datasets of different sizes. For example, methods such as minimum volume ellipsoid require all data before analysis is done, while decision trees and neural network classifiers need pre-classified data. When looking at scaling, most algorithms work well with a small dataset, while for example nearest neighbor and clustering algorithms are a suboptimal choice for handling large volumes of data (Hill and Minsker, 2010).

Rule-based approaches to anomaly detection are popular, especially within network surveillance, as rules can be added or modified without affecting other rules. They can be based on both normal and abnormal data, or only normal data. They can also be set up to compare sequences of actions. Approaches such as decision trees are adept at handling big, noisy datasets of high dimensionality, but they are dependent on the quality of the training data. They do not generalize well, and are susceptible to over-fitting (Hodge and Austin, 2004)

When selecting an algorithm to perform anomaly detection, it is important to consider if the algorithm models the data well and if it is scalable to the intended dataset (Hodge and Austin, 2004). Factors such as the processing power of the system and if the process should be done in real time or not should also be taken into account. Gunter et al. (2007) explore different techniques for detecting anomalies in large distributed systems, and argue for a light-weight approach to enable real-time detection. No single optimal method was found in their study, their conclusion being that several methods probably need to be combined as the effectiveness of different methods varies depending on the type of anomaly. Hodge and Austin (2004) agrees about the necessity of using a combination of methods to overcome weaknesses of one particular classification algorithm. Real-time detection is a feature that is highly desirable when dealing with system surveillance.

One further thing that can be done when performing anomaly detection is dividing data into different parts, and analyzing those parts separately. Lee et al. (2001) suggests doing this by for example dividing log files into classes based on the time of day, such as morning, noon, evening and night. This would for example be suitable for web traffic data that has different behavioral patterns during different times of the day.

### 2.3.1 Statistical anomaly detection

Statistical methods are suitable for quantitative data with few varying attributes, some methods such as Grubb's method are even only applicable for univariate data, which has a single attribute. The strict requirements on the data increase the need for preprocessing and transforming the data before statistical methods are used (Hodge and Austin, 2004).

The earliest approaches to anomaly detection were based on statistical methods (Hodge and Austin, 2004), and there are hundreds of outlier tests designed for different circumstances. The different tests take different approaches to the data distribution, whether parameters such as mean and variance are known, how many outliers are expected, and the type of outliers that are expected. It is a problem that many approaches are distribution based, as some of the distribution of the data may be unknown and thus lead to a need of extensive fitting. There are methods for handling distribution fitting in several dimensions, but they only have an acceptable performance for two or fewer dimensions (Knox and Ng, 1998).

Statistical methods are suitable both for static and dynamic data as it is possible to fine tune an existing model to new data. They can be as simple as using a box-and-whiskers chart to find outliers, or using features such as thresholds based on a number of standard deviations (Hodge and Austin, 2004). A more complex usage can be found in Veasey and Dodson (2014), where the authors argue that the p-value, which is a measurement concerning the rarity of a data point, can be used to rank them according to how unusual they are.

Statistical hypothesis testing is used to find anomalies, and can be used to assign a level of abnormality. Many methods rely on the simplicity assumption, that statistical rarity is proportional to the likelihood of a fault. This assumption is violated by for example systematic faults, which means that they cannot be detected by only using statistical anomaly detection (Srivastava and Srivastava, 2010).

Hodge and Austin (2004) divide statistical anomaly detection into four approaches, namely proximity-based, parametric, nonparametric and semiparametric. Proximity-based techniques have no inherent assumptions and are fairly simple. Examples include k-nearest neighbor, which uses Euclidean distance and majority voting to determine if a point is far enough away from all other points to be counted as an outlier, and k-means which tries to minimize the sum of squares in a cluster. Parametric methods are suitable for large datasets as the size of the model depends on the dimensionality of the data rather than the number of observations. Examples include least squares regression and convex peeling. Non-parametric methods are flexible and suited for data that do not follow a single distribution or follows a random distribution. There are also semi-parametric models that apply several local distributions to the data, rather than relying on a single global distribution (Hodge and Austin, 2004).

### 2.3.2 Supervised anomaly detection

Supervised anomaly detection requires pre-classified training data that defines normal behavior. Supervised algorithms may not be able to properly handle data from an unexpected region, as it was not contained in the training data (Hodge and Austin, 2004). Thus, it is important for the training data to cover as much of normal behavior as possible, including examples of normal and abnormal data, which may be difficult and resource draining to assemble. To do this, the different attributes of the data should be considered, if they are essential to the description of the data they should be included when training the model (Lee et al., 2001; Bolton and Hand, 2001).

One supervised method is the usage of classification rules, which consists of a set of rules that describe patterns related to important attributes of the data. The rules model both abnormality and normality (Lee et al., 2001).

### 2.3.3 Unsupervised anomaly detection

Unsupervised anomaly detection does not require any pre-classified training data or input from a human teacher, it learns about the normality of the data in an unsupervised manner. It is able to find outliers without having any prior knowledge of the data by processing a batch of data (Hodge and Austin, 2004). An advantage of unsupervised methods as compared to supervised methods is the possibility of detecting previously unknown types of faults (Bolton and Hand, 2001).

One popular unsupervised approach is clustering, a method that tries to segment data into groups. It can be used to find natural groupings and patterns in data. It is one of the most popular unsupervised techniques, but it can suffer from choosing data metrics poorly (Bolton and Hand, 2001). When used to detect malicious anomalies, for example during network surveillance, it relies of the observation that malicious events often are related, rather than occurring separately (Sample and Schaffer, 2013).

### 2.3.4 Neural networks

Neural networks are somewhat inspired by how a brain functions. They consist of a network of nodes that contribute to the computations. The neural networks are able to generalize to previously unseen patterns and can learn complex behavior. After training and testing has been done, they create a classifier to be applied to new data. One disadvantage is that most methods need to pass through the data several times to properly build a model, and must be tested to determine if threshold levels are set correctly. They are adapted to data with few different metrics, data of so called low dimensionality (Hodge and Austin, 2004).

Neural networks are often used to do predictive analysis. Applied to data in the form of for example network traffic, anomalous behavior can be explained and predicted (Sample and Schaffer, 2013).

There are both supervised and unsupervised approaches to neural networks. Supervised methods learn from the classification of the data, while unsupervised model the data by clustering provided training data into vectors. One example of unsupervised neural networks is self organizing maps (Hodge and Austin, 2004).

## 2.4 Information visualization

Information visualization is used to make data more comprehensible by illustrating it in some way, for example through bar charts or scatterplots. By visualizing the data, it is possible to quickly see patterns that would otherwise take a long time to find, or perhaps even remain hidden.

According to Chang and Ding (2005), figures are required to have a low dimension in order for humans to understand them. In fact, two dimensions are the optimal space for exploring patterns, and the act of adding even one additional dimension disables many pattern-finding mechanisms of the human brain (Ware, 2013).

While figures should preferably not exceed two dimensions of space, there are a number of visual dimensions that can be used to illustrate different kinds of information. These include for example position, color and size. If the information exceeds the number of applicable visual dimensions, one must use multiple visualizations. It is then important to create an apparent link between the different visualizations, so that the user can still understand it (Vesanto, 1999).

If the data to visualize is complex, it may be necessary to summarize it. This reduces the data that must be displayed, but once it has been summarized it may become difficult to work further with the data. It is also important to remember that there are different purposes to visualization, and the contents must be adapted to the intended audience (Chuvakin, 2013).

## 2.5 Log analysis

Currently, there are some tools that have functionality similar to what this master thesis will explore. For example, by using the tool *Logentries*, it is possible for the user to set up thresholds for specific alerts and values to dictate what is considered abnormal behavior. It also has some extended functionality such as inactivity alerts, which alerts the user when certain actions have not happened. However, there is no functionality for the system to, on its own, find erroneous behavior without previous human input. Visually, Logentries is very focused around how logs have been tagged, and presents the division of logs in those groups in a variety of ways (Logentries, 2015).

Another tool is *Google Analytics Intelligence Events*, which is focused on data generated by web traffic. It is able to automatically detect significant statistical changes in the usage data and traffic data of the user's website, and alert the user to such changes. However, it only monitors web traffic (Google, 2015), and is thus not a tool suitable for monitoring the system state but rather a tool for supervising and managing a web page.

A final example is the tool *Splunk*. It specializes in gathering machine data, and offers search of data as well as visualization and custom alerts. Splunk is able to automatically extract some fields from data, such as the source of for example HTTP-requests and timestamps. If the user wishes to further classify data, it has to be done manually. Roughly described, Splunk has a number of predefined fields, alerts and statistics, and any further functionality or data that does not fit those formats must be manually input (Splunk Inc., 2015).

A paper by Yamanishi and Maruyama (2005) explores the mining of *syslog*, a standard for the formatting of logs, for network failure monitoring. They base their model around the fact that the data is dynamic, which in turn leads to a dynamic model that should learn in real time. They use a series of hidden Markov models with dynamic learning of parameters, and gradually phase out old data from the model. They use of a score-based anomaly detection applied to sessions of data, checked against threshold values to determine if a session is anomalous. Yamanishi and Maruyama (2005) thus demonstrate the need to use several different techniques to account for different aspects of the process, such as how to learn and how to distinguish anomalous sessions.

# 3 Method of development

As a basis for comparison, two different approaches are explored. Insights gained from these approaches are then used to build the prototype, as described in Chapter 4. The master thesis work is then divided into three phases: basic implementation, extension, and evaluation. The first phase focuses on building a simple system model based on numeric values. This model is used for anomaly detection, and is visualized in a basic way. During the second phase, textual information is analyzed and clustered into categories. The model is extended with information about these, and more complex visualization is added. The third phase focuses on concluding the project and includes testing as well as evaluating the finished product.

Log files provided by the company 3bits Consulting AB are used to develop the prototype as well as to evaluate how well the prototype is able to reflect the behavior of the system and whether erroneous behavior is spotted. Log files containing errors as well as normal behavior are collected during the project. Log files from a Windows 8.1 machine and a Ubuntu 14.04 machine owned by the authors are also used during the evaluation, but not the development. Logs are assumed to be stored in a text format where each line in a log file represents one log entry, divided into one file per day.

As the thesis is based on the rather ill-defined problem of investigating if it is possible to determine the system state based on its log files, a substantial amount of time is dedicated to formulating a precise problem description and requirements for success. The resulting goals are listed in Section 1.1.

## 3.1 Domains investigated

This section describes the domain of the test data, namely the e-commerce system managed by 3bits Consulting AB, also known as 3bits, as well as the Windows and Ubuntu log files. First, the current method of handling log files at 3bits is briefly described. The rest of the section gives a description of the amount and format of log files from

3bits, Windows and Ubuntu. Though data from these particular systems will be used in the development and testing of the prototype, the project will take an unsupervised approach in order for the prototype to be applicable on different systems.

### 3.1.1 The log managing process of 3bits Consulting AB

The current method of handling log files within 3bits Consulting AB is rather reactive. In line with the statement by Jayathilake (2012), the log files are manually analyzed. This is done in order to find the cause of an already known error. For example, if a feature is reported to be malfunctioning, the log files are accessed and an employee manually goes through the relevant files to find the source of the error. The log files are also utilized by inspecting their sizes. If the sizes are found to deviate too much from the normal state, an employee manually investigates the cause of the change.

### 3.1.2 Data from 3bits Consulting AB

Of the many log files produced by a system run by 3bits Consulting AB, a total of 52 different log files are used during the development and testing of this project. These are all daily, informative log files with static names, which enables comparison on a file level. Two of these log files are generated at a web server, while the other 50 are found on an internal server. In general, the files on the internal server take up around 6-8 GB in total per day, and are stored for around a month. In comparison, the two files on the web server add up to 2-3 GB per day and are only stored for one week before deletion. The data used to test this thesis was collected between February and May 2015 for internal data, and March and May 2015 for the web server data. In total, around 4500 log files were collected, consisting of roughly 314 GB of data.

Most of the log files that are used as data are of the format seen in Listing 3.1. Each log row consists of a timestamp, the name of the process producing the log, level, and message. Though conforming to this structure, some log files differ slightly in the naming of the process. In them, the process name is encapsulated in tags and is in some files simply an increasing number as seen in Listing 3.2. In other files, the process name is followed by a colon and a number which likely symbolizes the thread ID. This format is seen in Listing 3.3. In these files, the process name may also be missing, leaving only the number on the right side of the colon.

The messages are written in English as a rule, but there are a few instances of Swedish logs. Information such as addresses and names are written in a variety of languages.

**Listing 3.1:** A typical log row

```
01:02:03.456 ExampleProcess 6: Starting process 'HelloWorld'...
```

**Listing 3.2:** Log row with encapsulated process name

```
01:02:03.456 <123456789>    6: Starting process 'HelloWorld'...
```

**Listing 3.3:** Log row with process name and thread ID

```
01:02:03.456 <Name : 123>    6: Starting process 'HelloWorld'...
```

### 3.1.3 Data from private machines

System and program log files collected from a personal Windows 8.1 machine are mainly used to test the flexibility of the prototype. The log files are in Swedish and span from August 2014 to April 2015, and contain data from 227 different days. An example of the format of the log files can be seen in Listing 3.4. Note that different parts of the log file are separated by tabs.

Furthermore, auth, kernel, and syslog log files are collected from a Ubuntu 14.04 machine for the same purpose. The log files are in English and span from January to April 2015, containing data from 30 days. An example of the formatting can be seen in Listing 3.5.

Since the log files are stored in a single file for each type, they are preprocessed and split into one file per day before they are run through the prototype. This also removes the date from the beginning of the line, but except for this no further changes are done to the log.

**Listing 3.4:** Log row from Windows 8.1 system log

```
2014-05-25 17:38:59    Information    Microsoft-Windows-Eventlog
104    Rensa logg    Loggfilen Application har rensats.
```

**Listing 3.5:** Log row from Ubuntu 14.04 auth log

```
Feb 10 11:13:28 j-Aspire-3619 pkexec: session opened for user...
```

## 3.2 Evaluation

The project is evaluated using the log files provided by 3bits, as well as log files from a Windows 8.1 machine and a Ubuntu 14.04 machine. Using data for an e-commerce system as well as data from private machines tests the flexibility of the anomaly detection as well as the tolerance for different formats. The evaluation involves checking the accuracy of the anomaly detection, the prototype's adjustment to any shift in the "normal" state, the amount of storage space required by the prototype, the efficiency of the visualization, and the flexibility in handling different log formats. During the evaluation, a Windows Server 2012 R2 with 4 GB of RAM is used to perform the processing.

The usability of the prototype is evaluated through a small number of user tests performed by persons who normally work with 3bits' log files. The most important issue to investigate is whether the prototype can be used to support log handling, if the user could use the prototype to solve or assist in solving existing problems. Other points of interest include whether any functionality is missing, and possible areas of usage. These tests will also evaluate the visualization by investigating whether the interface seems intuitive to the user. This includes whether the user is able to navigate the interface and spot the anomalies. To do this, the users are asked to perform a number of tasks and are then allowed to explore the prototype on their own. Any additional comments from the users are also noted.

To evaluate the effectiveness of the anomaly detection, the anomalies discovered by the prototype are compared to the contents of the log files. When the detected anomaly corresponds to an anomalous event in the log file, it is considered successful. Instances of known anomalies that are not detected are considered as failures. In order to have data with known anomalies, 3bits' system is monitored during the development so that actual errors are known. It is of interest to know why an anomaly was not detected in order to be able to make future improvements, as well as why any false positives occurred.

Closely linked to the anomaly detection is the normal state of the system. If the prototype is not able to adjust well to changes in normality, there is a great risk that the results of the anomaly detection will be affected. Any errors in the anomaly detection stage might therefore originate from a bad interpretation of the normal state. It is thus important to check with what efficiency the prototype can adjust to changes in normality and to keep the results from this evaluation in mind when performing the evaluation of the anomaly detection stage.

As for the storage space used by the prototype, this is compared to the amount of space needed to store the original log files. Worth keeping in mind is that the data stored by the prototype is aggregated and intended to date back further than the original files.

# 4

# Two approaches to implementation

Although the project mainly uses log files from 3bits' system for testing, the prototype should ideally be more generic and applicable on logs which use other formats. As an examination of different methods and as a basis for comparison, two different approaches are investigated: one specific approach focusing fully on 3bits' log format, and one generic approach that could be applied to any row-based plain-text log format. The experiment is conducted in order to find positive and negative aspects with the two approaches in order to make the final product as useful as possible.

## 4.1 A specific approach

By focusing only on the format of the log files from 3bits, a log row may be split into a timestamp, the name of the process producing the log, a level, and a message as described in Section 3.1.2. This easily reveals the time, process and level of the log, and logs can then be grouped based on one of these attributes. The message can also be extracted, although the complexity of the message varies and in many cases there cannot be any immediate grouping. Many messages contain variables like product ID, and so even though the stem of two messages are the same, they might not be equal.

To solve the problem of grouping messages, a simple method for replacing suspected variables is created. Using *regex* (regular expressions), messages are converted into *stem messages*. The message is split into *tokens*, which are here defined as strings of any characters but whitespaces. Tokens consisting only of the lower-case letters *a-z* with one or no prepending upper-case letter *A-Z* are considered part of the stem and are thus kept as they are. Tokens that consist only of special characters are also unchanged. Any special character in the beginning or end of a token is treated separately from the rest of the token. This allows tokens that, for example, end with a comma to be recognized as part of the stem. All other tokens are then replaced with a *\S**, regex for “zero or more characters of any sort but whitespaces”.

This simple regex method manages to find many stem messages, though there are still many messages for which the method is less effective. One example is messages containing SQL statements which vary in length. Another is messages with variables that are expressed as ordinary words. Both cases lead to multiple stem messages being created. Since the specific approach only takes messages of this particular system into account, the problem can be solved by creating exceptions for messages that are known to start with, be equal to, or contain a specific string. Certainly, this is not the most gracious way to handle the situation, but it is effective enough for a simple model.

Processes and levels can be associated with several different messages, but a message is most likely always connected to a specific level and the specific process that produced it. Thus, while grouping logs on process and level can give the user an idea of the combined status of all logs associated with that process or level, grouping on message provides the user with much more specific information. This makes the message the most interesting of the three. As a message is connected to a level and process, these three can be grouped together as one *key*. The key enables a new level of detail. Normal behavior is based on the number of logs produced per day and can now be defined on three levels: the number of logs produced in total, the number of lines produced per log file, and the number of lines produced per key.

### 4.1.1 Normal behavior and anomaly detection

For this simple model, log lines are summarized by minute and hour. A *weighted average* is used to define normality, together with a weighted maximum and a weighted minimum. At first, the weighted average $avg_{n+1}$ for the next day $n+1$ is calculated like a normal average based on all existing summarized log files $day_i$, as seen in Equation 4.1. Once a weighted average $avg_n$ for the day $n$ exists, it is used together with the latest log file $day_n$ to calculate $avg_{n+1}$ with Equation 4.2. This is done in order to allow the model to evolve with new data, as data from the last month is considered more relevant than data from a year ago. The constant $k$ can be used to modify the impact that $avg_n$ should have on $avg_{n+1}$. This approach uses $k = 1$, which means that $avg_n$ and $day_n$ will have equal impact on $avg_{n+1}$.

$$avg_{n+1} = \frac{\sum_{i=0}^{n} day_i}{n+1} \tag{4.1}$$

$$avg_{n+1} = \frac{k * avg_n + day_n}{k+1} \tag{4.2}$$

The weighted maximum and minimum are calculated in a similar way to the weighted average. This is done since simply keeping the most extreme value would affect the model forever, and prevent adaptation. The difference from the calculation of the weighted average is that the maximum $max_{n+1}$ and minimum $min_{n+1}$ are weighted more towards the most extreme value of $day_n$ and $max_{n+1}$ or $min_{n+1}$, as seen in Equation 4.3. Here,

$k = 9$ is used to give the most extreme value 90% of the impact on the maximum and minimum values of the next day.

$$\begin{aligned} max_{n+1} &= \begin{cases} \frac{k*day_n+max_n}{k+1}, & if day_n > max_n \\ \frac{day_n+k*max_n}{k+1}, & otherwise \end{cases} \\ min_{n+1} &= \begin{cases} \frac{k*day_n+min_n}{k+1}, & if day_n < min_n \\ \frac{day_n+k*min_n}{k+1}, & otherwise \end{cases} \end{aligned} \tag{4.3}$$

The reason for using maximum and minimum values instead of, for example, the standard deviation is to get a more correct picture of the normality. Many log files have a much greater difference between maximum value and average value than they do between minimum value and average value, and vice versa. Using standard deviation would level recurring peaks that are necessary to avoid classifying new peaks as anomalies. It would also allow for extra deviation in the opposite direction, thus allowing for anomalies to slip through.

As the maximum value is never as high as the latest peak, the threshold needs to be extended to some degree in order for a peak to reoccur without being labeled an anomaly. Equation 4.4 shows how, in this model, the distance between the maximum value $max$ and the average value $avg$ is doubled and then used as a threshold $t_{max}$ for anomaly detection. Any values beyond this threshold are considered anomalies. The same is true for the minimum value.

$$\begin{aligned} t_{max} &= avg + 2(max - avg) \\ t_{min} &= avg - 2(avg - min) \end{aligned} \tag{4.4}$$

The results are displayed in a line diagram showing one day at a time. As illustrated in Figure 4.1, the black line represents the values from the date selected while the green line represents the weighted average for that date. The green area is created from the weighted maximum and minimum values, and symbolizes the safe area. Values within this area are treated as normal, while outliers are labeled anomalies and marked with red dots. Thus, the function of the average line is to provide the user with more information regarding how normal the values are upon viewing.

## 4.2 A generic approach

As the generic approach strives to make as few assumptions as possible about how logs are formatted and stored, two simple assumptions are made. The first assumption is that the logs are line-based. The second assumption is that it is possible to divide the logs into files based on what hour the logs were produced, either at runtime or by some contained timestamp. The logs, divided by hour, are then analyzed to find abnormalities. The period of the analyzed files refers to the days the log files span.

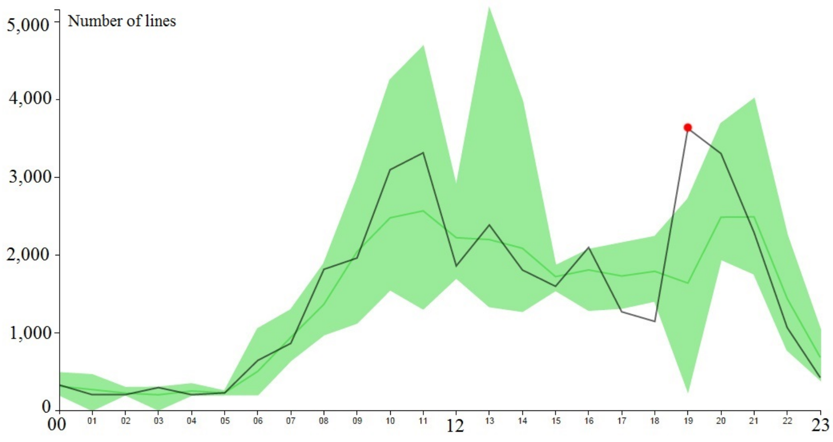


**Figure 4.1:** Model of lines over time with maximum and minimum values, slightly modified for readability.

### 4.2.1 Counting lines, words and characters

One rough estimation of how much activity there is in the log files for a given hour can be found by counting the number of output lines. Though different log files may output different amounts of logs, the total number should be fairly constant during normal usage. We define $L(h)$ as a count of lines in log files during hour $h$.

One other simple measure of measuring the system state is counting the total number of words, as well as how often each specific word occurs. All words found in the log files are assembled into a matrix $W(w,\ h)$, where each entry represents how often a word appears in a specific hour.

To be able to compare word $W$ matrices for different periods, they are normalized into normalized word matrices $WN(w,\ h)$. For each word, its proportion of the total word count for each hour over the period is compiled. This is done using Equation 4.5. Normalization allows comparison between a long period of time and a single day, and the data can then be used to find how normal that single day is when using the period as a measure.

$$WN(word, hour) = \frac{W(word, hour)}{\sum_w W(w, hour)} \tag{4.5}$$

As an extension to counting lines and words contained in the log files, it is also possible to count the the number of characters therein. It stands to reason that the number of lines, words and characters are related, and as can be seen in Figure 4.2, Figure 4.3 and Figure 4.4 all three counts are strongly related. All three figures plot the count of a metric against time, and while the curves for word count and character count are nearly identical, the curve for line count is slightly rounder in its curves, most notably so around 05:00. Since characters do not provide any new information that can not be found in the word counts, the character count will not be collected or used.

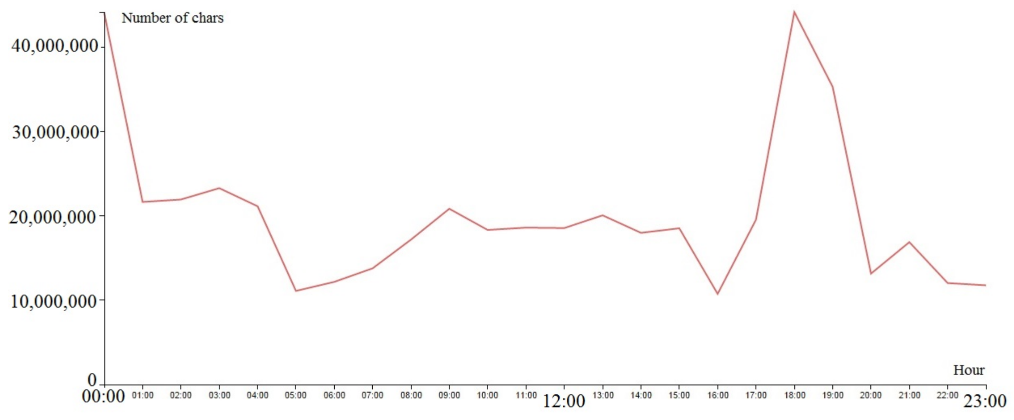


**Figure 4.2:** Number of characters plotted against time, slightly modified for readability.

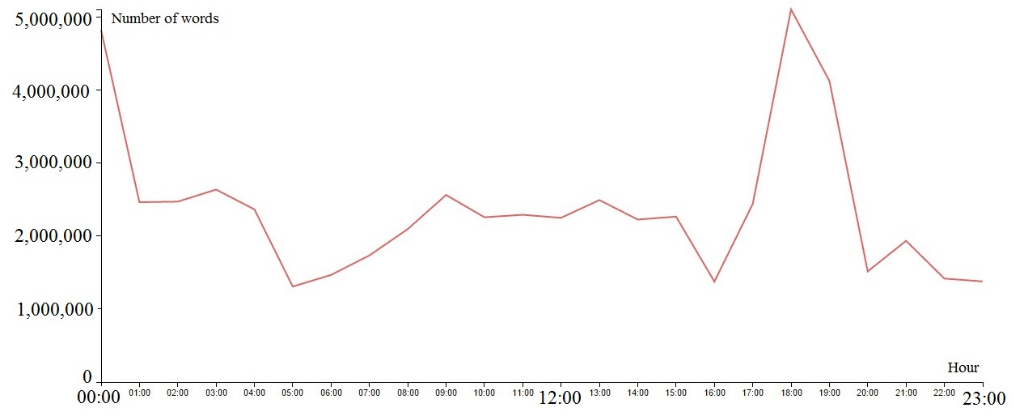


**Figure 4.3:** Number of words plotted against time, slightly modified for readability.

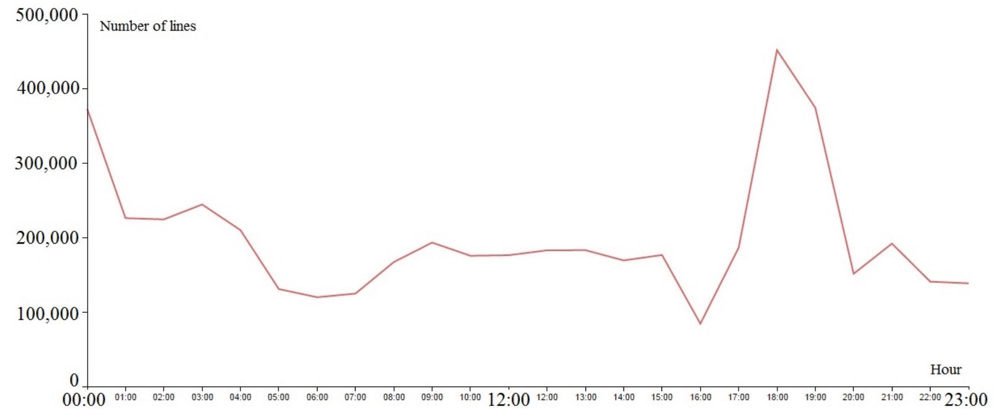


**Figure 4.4:** Number of lines plotted against time, slightly modified for readability.

### 4.2.2 Word bigrams

N-grams are a measure of a sequence of length *n*. They are one way to investigate the composition of a text, and are used here to gain an understanding of what the log files contain. A subsection of n-grams are *bigrams*, n-grams where $n = 2$. For the sentence "This is a test", the list of bigrams would be [This is, is a, a test]. For this thesis, bigrams of the words in the log files were found and put into a bigram matrix *B(bigram, hour)*, where each entry is the count for a bigram and an hour.

As with word matrix *W*, bigram matrix *B* is normalized into normalized bigram matrix *BN(b, h)*, where each count represents the proportion a bigram makes up out of the total number of bigrams found over the period for that hour. The normalization is done in the same manner as for *WN*, as can be seen in Equation 4.6.

$$BN(bigram, hour) = \frac{B(bigram, hour)}{\sum_b B(b, hour)} \tag{4.6}$$

### 4.2.3 Grouping on message

To find logs that are of the same type without making any assumptions about what is contained in the logs, a number of methods can be used. One such method is comparing two logs based on Jaccard distance, which looks at the common letters in the two logs. Due to the length of the logs, most are longer than 100 characters, the probability of containing the same letters were high and logs that were not similar were grouped together. Another possible measure is Levenshtein distance. It finds the number of necessary edits to convert one string into another. The resulting groups are much better than those produced by the Jaccard distance, but the comparison is very slow. The absolutely best grouping is produced by looking at the longest common substring of the logs, but it is also the slowest.

In general, the more specific and precise the grouping is, the longer the comparison take to perform. This is partly due to the comparison itself, and partly to that more exact grouping yields more groups that must be tested for membership. For example, calculating the Levenshtein distance for 1,000 lines takes 2.6 seconds while finding the longest common substring takes 10 seconds. This indicates that at peak production there is a possibility of one hour of logs taking more than one hour to analyze when using the more exact method, since it is not uncommon for the logs from 3bits to span for more than 400,000 lines.

### 4.2.4 Normal behavior and anomaly detection

To assess the normality of, for example, a day of logs, statistical measures such as mean and standard deviation are applied to the count of lines, total words and total bigrams. A deviation of more than two standard deviations from the mean is considered an anomaly. Counts for specific words and bigrams are monitored based on their change measured in percent, as it is common that the count for a certain word increased by several 100%,

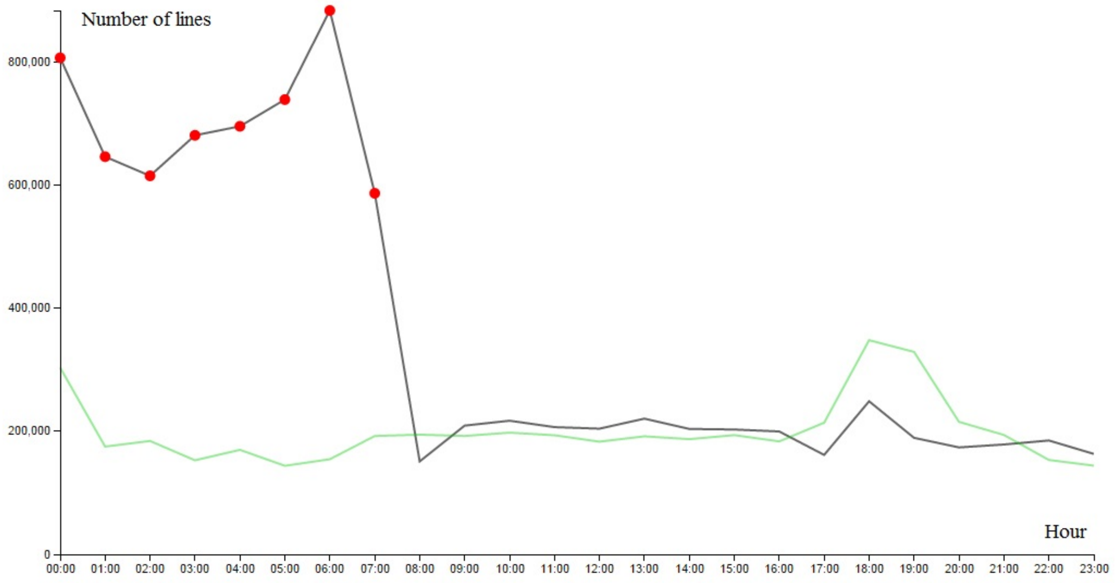


**Figure 4.5:** The visualization of number of lines plotted against time, including anomaly detection

while a decrease rarely is more drastic than 50%. The top ten words for each hour are found and printed to a file in order to roughly describe the contents of the log files.

Anomalies have been visualized as seen in Figure 4.5. The figure contains line count plotted against time, and abnormal points are marked by red dots as can be seen in during 00:00-07:00. A green line marks the usual behavior of the metric.

# 5

# Unified approach

The unified approach combines aspects from both Section 4.1 and Section 4.2 in an attempt to make a usable prototype that is generic enough to run on more than a single system. This chapter briefly describes some general points learned from the preliminary approaches, and then describes different decisions made regarding the final, unified approach. It is divided into sections roughly corresponding to the flow of the program.

After the development of the two separate approaches, an insight was gained about the necessity of making assumptions about the format of the data. Though the generic approach was able to find anomalies and give a brief view of the system, it could not provide a lot of information that would have been useful for users, such as where in the system the abnormality occurred. The counts of the total number of lines, words and bigrams were useful, but too generic to be able to assist the users in locating and removing errors. The contents of the logs are useful for understanding the system, so they should be included by for example visualizing counts of specific words. In order to be able to divide the logs by origin, not hour, it was found necessary to assume that each log row begins with a timestamp of some form.

To ensure that historical data is always included, it is important to perform incremental analysis. By doing this, it is possible to gradually forget old information and adapt to a changing environment, as was found in the specific approach. Furthermore, to accurately describe the uneven distances to the average, a combination of standard deviation and the weighted average from the specific approach is used.

## 5.1 Granularity of data

In the case of log files there are a number of levels of granularity of data available, ranging from milliseconds to years. Increased granularity leads to longer processing time and memory usage, but also provides more information to analyze. The question of what granularity to assume is not an entirely trivial one, as exactness of analysis is weighed against speed of processing. Too fine granularity may also lead to the user being overwhelmed by the amount of information available.

The granularity of this project is that all calculations are done on an hourly basis. It was found during the separate approaches that performing analysis based on seconds or minutes was very noisy and demanding. Reports being logged at 15:03 one day and 15:11 the next are not uncommon. Thus, statistics based on minutes become fairly unreliable and require a very long period of data to be useful. The granularity of an hour gave a good overview to the system whilst providing enough interesting information.

Metrics of the data are thus calculated per hour. The metrics are compared to the behavior of the corresponding weekday; logs that are produced during a Monday will be compared to previous Mondays. The metrics are then skewed in order to take the general trend of the week into account, as described in Section 5.3. The weekday comparisons are done due to the cyclic behavior based on the weekdays, something that is especially visible in an e-commerce system. Due to the scope of this project, and the availability of test data, comparisons are not done on a monthly basis though there is likely to exist a cyclic behavior over the months as well.

## 5.2 Preprocessing log files

In order to effectively analyze the log files, they are first preprocessed. For each log file, a number of summaries of an aspect of that log file are compiled. Among these are the counts for the total number of lines, the total number of words and the total number of bigrams. All log rows in the file are analyzed and sorted by their contents into so called *message groups*, as described in Section 5.2.1. For each word, bigram, and message group that is found during preprocessing, the number of occurrences per hour and what percentage of the file that consists of that word, bigram or group is collected. The counts are collected in the manner described in Section 4.2, with the exception that units that only occur once are not included (following the reasoning in Section 5.3.1).

These summaries are used when analyzing the log files, due to providing easy access to information that can be used to judge the state of the system. They are also used when visualizing the results. After the log files have been preprocessed, they can be discarded as they are not used again by the prototype.

### 5.2.1 Grouping messages

In order to group log rows based on their content, their similarity is judged based on the length of their longest common substring, as this was found to produce the best results in Section 4.2. To do this, the C# library *FuzzyString* is used (Jones, 2013). Each log file is grouped in this manner, and the relative frequency of messages of the different groups is calculated.

Though Hamming distance was a candidate for determining the similarity of strings, it was rejected as the strings often are of unequal length. During testing, it was also found to be slower than longest common substring, and produce approximately the same results. Other measures such as Jaccard distance were also rejected during the generic approach.

During the specific approach, regular expressions were used to group log rows. This technique is not used in the unified approach due to the difficulty in automatically creating regular expressions that could handle variables in a log line. During the specific approach, a list of specific special cases was used, which is not possible when the contents of the log files are unknown.

## 5.3 Analyzing the preprocessed files

Once the log files are of an appropriate format, several statistics are calculated concerning them. For all counts of total lines, total words and total bigrams hourly maximum and minimum values are found, as well as the average and standard deviation.

If no analysis has been done previously, several days of preprocessed log files are input in order to calculate the metrics mentioned above. The number of required days is variable, but seven days are used by default. If a previously done analysis is available, the latest computed analysis is combined with the new information from the new day in order to gradually update the model of normality. Due to weekly patterns noted in the data, see Section 5.1, this is done on a weekday basis.

An example of how an existing analysis file is combined with daily values can be seen in Equation 5.1. By default, the historical and new data are treated as having the same worth, so the constant $w$ is set to 1. The variable $l$ represents the line count, while the variable $a$ is the average value computed from the historical data.

$$avgLines(l, a) = \frac{a + (l * w)}{w + 1} \tag{5.1}$$

Maximum and minimum values are also computed by taking a weighted average, though they are more skewed towards the most extreme value, as can be seen in Equation 5.2. The constant $w$ is set to 9 by default in this case, to place more emphasis on the extreme values. The variable $m$ contains the maximum value of the data, while $n$ is the maximum value of the historical data. The weighted averages have the advantage of considering historical data, and slowly forgetting it as it becomes outdated. Based on the calculated averages, the standard deviation is calculated.

$$maxLines(m, n) = \begin{cases} \frac{(m*w)+n}{w+1}, & if\ m > n \\ \frac{m+(w*n)}{w+1}, & otherwise \end{cases} \tag{5.2}$$

In order to take any overall changes into account, the maximum and minimum thresholds are skewed with respect to any change noted during the last three days. This is done through linear regression by using the ordinary least squares method as seen in Equation 5.3. For every hour of the last three days of the week, the change from last week is derived. The change is then used as input in the equation. For example, if the day is a Thursday, the change in hour 00 could be calculated as in Table 5.1. This would result in the values $y_1 = 1{,}000, y_2 = 2{,}000, y_3 = 5{,}000$ ($x$ is simply calculated as $x_i = i$, giving us $x_1 = 1, x_2 = 2, x_3 = 3$). The equation is used to find $y_4$, the probable change for today. If $y_4$ is positive, the change is added to the maximum value, and if it is negative, the minimum value is the one to be altered. In this way, the skewing can only expand the acceptance range, never diminish it. This allows for quick adaptation to overall changes in the numbers without yielding any anomalies due to false alarms.

$$\begin{aligned} y &= \alpha + \beta x \\ \beta &= \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} \\ \alpha &= \bar{y} - \beta\,\bar{x} \end{aligned} \tag{5.3}$$

**Table 5.1:** Change of values in hour 00 over three days

| Weekday | Last week | This week | Change ($y_i$) |
|---|---|---|---|
| Monday | 3,000 | 4,000 | 1,000 |
| Tuesday | 5,000 | 7,000 | 2,000 |
| Wednesday | 4,000 | 9,000 | 5,000 |

The occurrences of all words, bigrams, and message groups are during analysis compiled into one table for each day and log file, containing the word, the number of occurrences for that word per hour, and the standard deviation for that word for the day. The importance of the word is calculated as described in Section 5.3.1.

The table is then combined with the table from the previous day in order to produce a table of average counts, as for the total counts described above. Due to the high variability of occurring words, an incremental average would include many words that have not occurred in a relevant period of time. To combat this, words with counts lower than two are not included in the table.

Due to feedback from individuals working within 3bits, the list of the ten most frequent words described in Section 4.2.4 was excluded from the unified approach. The information it yielded was not useful enough to justify the additional processing time.

### 5.3.1 Calculating the importance of a word

A word is important, or at least the norm, if it occurs often. Rarely occurring words are interesting as well, but they are very common within 3bits' log files. Most words occur very few times, most likely due to being variable names or similar. For example, out of 200 different words produced by one log file during a single hour, the top 14% of the words made up 60% of the total file. Similarly, for the same number of bigrams for the same file and hour, the top 35% of the bigrams made up 90% of the file. This division

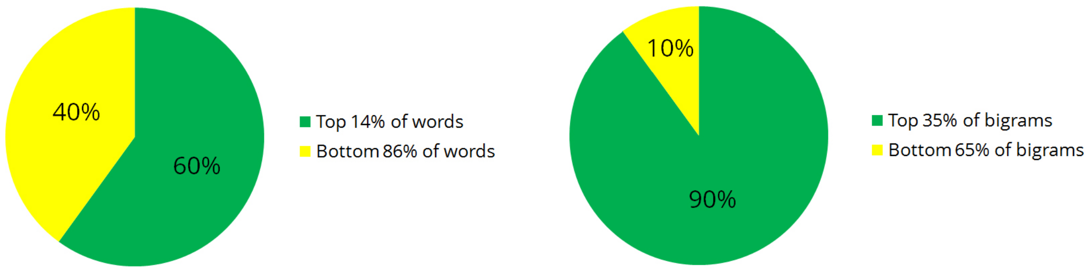


**Figure 5.1:** Contents of a log file divided by most and least common words and bigrams

can be seen in Figure 5.1, where the division of words is to the left while the bigrams are to the right.

The sparsity of words is further supported when considering the bigram matrix $B$ and the word matrix $W$ compiled during the generic approach, as they were found to be sparse. The majority of the words in $W$ occurred a low number of times, usually during a single or a few hours. An example of parts of a few lines from $W$ can be found in Table 5.2, where the word "exporting" is an often occurring word while "framework" and "1762173" are not. Once $W$ was normalized, it became obvious that most words made up less than 0.01% of the total, following the pattern seen in the previous paragraph. A few words made up the bulk of the file.

**Table 5.2:** Excerpt from $W$ for a period of seven days, hour 00-09

| Word | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
|---|---|---|---|---|---|---|---|---|---|---|
| exporting | 1,524 | 1,009 | 933 | 910 | 1,032 | 929 | 1,457 | 2,280 | 4,159 | 6,262 |
| framework | 298 | 27 | 7 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| 1762173. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

Because of this large influx in words, looking at new or seldom occurring words will not provide a lot of useful information. However, if a top word disappears or a word that normally occurs during a single hour is found during a different hour, it may signify an important change. It is thus interesting to monitor such a drop or change over time.

To include these aspects when calculating the importance of a word $IW$, Equation 5.4 is used. $TW(h)$ is the total number of words in a log file during hour $h$, while $CW(w, h)$ is the count of the occurrences of a specific word $w$ in a log file during hour $h$. $OC(w)$ is the number of hours the word $w$ occurs in.

$$IW(w) = \begin{cases} 0, & if\ OC(w) = 0 \\ \sum_{i=0}^{23} \frac{CW(w,i)}{TW(i)} * \frac{1}{OC(w)}, & otherwise \end{cases} \tag{5.4}$$

The importance of a word could be used to help assist the visualization of words and bigrams. A higher importance would lead to the word being prioritized if there is limited space for visualization. This is one way of preventing a user from being overwhelmed by information about irrelevant words. Due to time limitations, frequency was instead used to assist the visualization.

## 5.4 Performing anomaly detection

Once the data has been analyzed, the files from the current day are investigated for abnormal points. The standard deviation as well as the skewed maximum and minimum values found during analysis are used to do so for the counts of the total number of lines, words and bigrams. As observed during the specific approach, only using the distance between the average and maximum value to set the threshold resulted in a very high threshold when the distance was high, and worse, a barely noticeable threshold when the distance was minimal. Thus, in line with Equation 5.5, the maximum and minimum values $max_h$ and $min_h$ are instead combined with the standard deviation $\sigma_h$ for the hour $h$ to level the differences in high and low distances. A point is considered an anomaly if it has a greater value than the maximum threshold $t_{max_h}$ or a lower value than the minimum threshold $t_{min_h}$.

$$\begin{aligned} t_{max_h} &= max_h + \sigma_h \\ t_{min_h} &= min_h - \sigma_h \end{aligned} \tag{5.5}$$

To find abnormal changes in occurrences for singular words, bigrams, and message groups, the average value for that hour is used together with the standard deviation for the day. To receive a greater threshold, the standard deviation $\sigma$ is multiplied with a constant $k$ before being added to or removed from the average value $avg_h$ for the hour $h$, as seen in Equation 5.6. If a value from the current day is greater than the maximum threshold $t_{max_h}$ or lower than the minimum threshold $t_{min_h}$, it is considered an anomaly. Through testing, $k = 3$ was determined to be a good level for the variance in the test files.

$$\begin{aligned} t_{max_h} &= avg_h + k\,\sigma \\ t_{min_h} &= avg_h - k\,\sigma \end{aligned} \tag{5.6}$$

If a word, bigram, or message group exists in the current day's file but not in the history file, that means it is new. New occurrences are not treated as anomalies since they are very common within the test files and would generate too much noise. However, old occurrences that suddenly disappear from a file can be labeled as anomalies unless their standard deviation is great enough, as they are treated as if their hourly values are zero.

As a rudimentary service, it is also possible for the user to specify words and bigrams that should always raise an anomaly. This is done in a separate file, the so called blacklist.

During preprocessing, any entry that is on the blacklist is noted and the occurrences written to a separate anomaly file. This is done during preprocessing to minimize the number of times the word and bigram lists are read, as this is one of the more taxing tasks of the prototype.

## 5.5 Visualization

The framework *D3* or *d3.js* is a popular JavaScript library for visualizing data (Bostock, 2012). It is used in this thesis to visualize the log files in different ways, as well as highlight anomalous behavior. Since files from all stages of processing are used during the visualization, all relevant files must be formatted in a way that is fit for inputting into D3. In the end, the data will be handled in a JSON format, but D3 also provides functions for parsing files with comma-separated values and values with other separators. For this project, tab-separated values files are used as they are more space efficient than JSON files. A tab is used as a separator since any tab in a message can be replaced by another white-space character without the message losing its meaning, whilst the same cannot be said about, for example, a comma.

Visualization is done on a web page using AJAX, which allows the content of the page to be updated dynamically without having to reload the page. While JavaScript is used on the client side mainly to visualize the data, PHP is used on the server side to handle any request the client might have. For example, when first opening the web page, the client asks the server for the latest date. It also asks for the list of log files, and lastly for the files with visual data that are connected to the first file in the list, which is automatically selected.

In Chapter 4, it was found that showing a single day of information did not provide a good enough context to understand the changes in content. Thus, in addition to being able to view each log file during a day, a view spanning seven days back in time is available. This allows the user to easily place the current data into a context. Though it is possible to extend the weekly view into a monthly and a yearly view, such views are not included due to the limited amount of data collected.

# 6 Results

This chapter describes the results of this project, specifically of the unified approach. The results in this chapter were obtained by running the prototype on log files provided by the company 3bits Consulting AB, as well as system log files from a Windows 8.1 machine and a Ubuntu 14.04 machine in order to test the flexibility of the prototype. Focus is placed on the log files from 3bits Consulting AB, as they are from a more complex system.

The assumptions made regarding the input data are that each line starts with a timestamp (as stated in Chapter 5), that the log files are line-based, and that there is a separate log file for each day (as stated in Section 3.1). All results are based on that these assumptions hold when treating the data.

The final version of the prototype consists of preprocessing that counts totals as well as individual occurrences of lines, words, bigrams, and similar messages. During analysis, minimum and maximum values of different counts as well as the average and standard deviation are found. Based on standard deviation, minimum, and maximum values, the prototype finds anomalies. The results are visualized as can be seen in Section 6.1.

The rough structure of data processing can be seen in Figure 6.1. Once the log files have been collected in one place some preprocessing is done on the files, which produces a number of files containing for example counts per hour and file. Thereafter, the files are analyzed, which includes calculating the average counts and standard deviation per hour and file. Finally, anomaly detection is applied, which first calculates the limits for acceptable values and finds unacceptable values. Any data that is found to be anomalous is stored in a separate file. It is important to note that all steps can be done for a single hour, which enables gradual analysis of data during the day.

To allow flexibility in how the log lines are formatted as well as where they are stored, there is a settings file, as seen in Listing 6.1. This file contains a number of settings pertaining to the location of different files and the format of for example timestamps. Before any processing is done, this file is read and the settings are loaded.

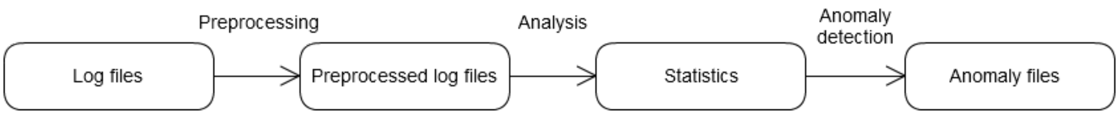


**Figure 6.1:** The different stages of processing

**Listing 6.1:** An example settings file.

```
-- Folders
startFolder	D:\LogFiles
preprocessedFolder	D:\PreProcessed
statisticsFolder	D:\Statistics
anomaliesFolder	D:\Anomalies
-- Formats of time and date
timeFormat	^\d{2}[:]\d{2}[:]\d{2}[.]\d{3}$
dateFormat	yyyyMMdd
-- Location of blacklist
blacklist	D:\UserInput\blacklist.txt
-- Default length of period
lengthPeriod	7
```

## 6.1 Visualization

The visualization consists of a single view with several components. These components display a graph over total counts, the occurrence of blacklisted words or bigrams, visualization of words and bigrams, and a listing over the anomalous message groups in the log files. To traverse between the different parts, the user scrolls up and down in the window. To change what file is displayed, the list to the right in Figure 6.2 is used. The log names have been blurred to protect the confidentiality of 3bits' data. To view data from a different day, the arrows next to the date can be used to browse between different dates. The double arrows change the date by a week.

In Figure 6.2, the total line count is displayed for all hours of a single day. The black line represents the actual line count for the day, while the dark green line represents the average of previous Tuesdays, since the picture is compiled from data gathered on a Tuesday. The green area maps the counts that are between the historical maximum and minimum values, which does not mean that a value has to lie within that area to be normal. Any anomalies are marked by a red dot, as can be seen at hour 11. Though Figure 6.2 counts the total number of lines, the same type of graph is available for the total number of words and bigrams. These can be accessed by clicking the buttons above the graph.

To view the data over a week instead of for a single day, the button to the right above the graph can be pressed to display the view in Figure 6.3. Except for spanning over a greater number of days, this view is the same as the one in Figure 6.2, though

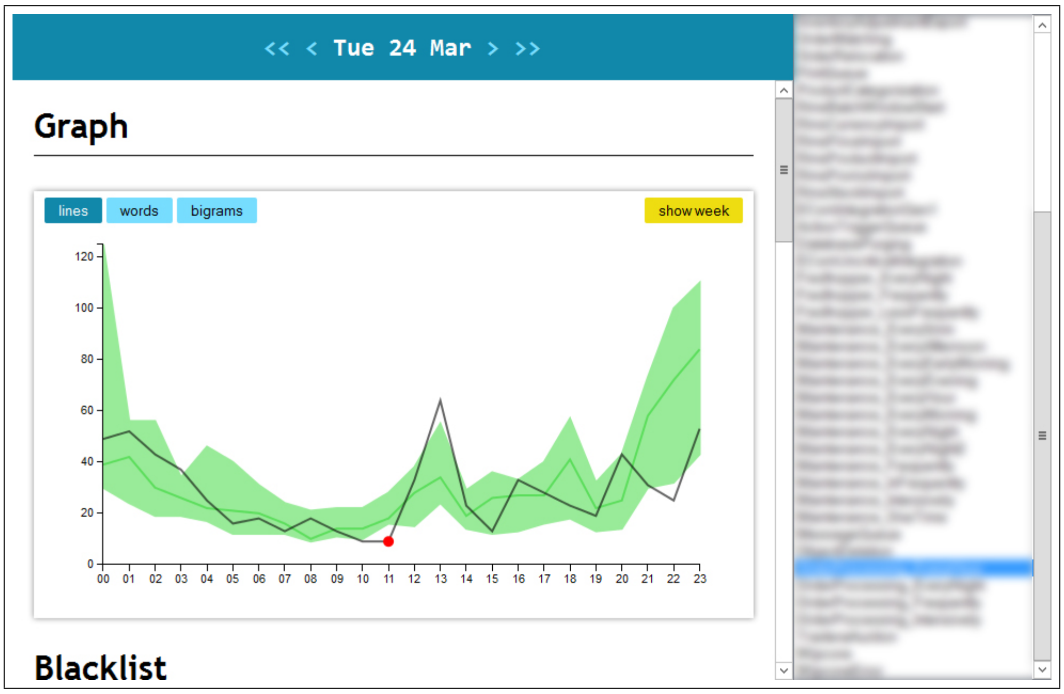


**Figure 6.2:** The interface for the prototype, including the total number of lines plotted against time for the duration of one day

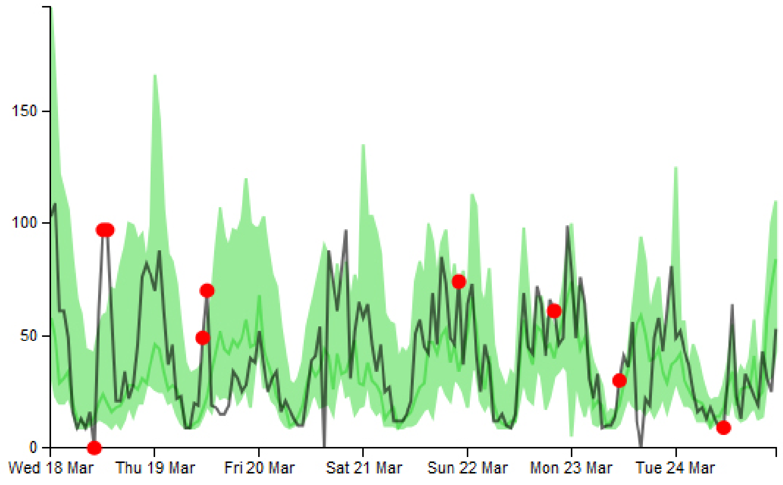
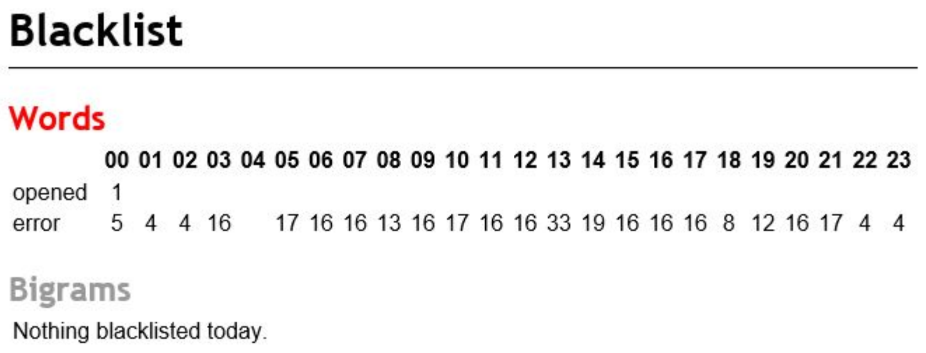


**Figure 6.3:** The total number of lines plotted against time for the duration of one week

the interface has been excluded from the figure.

Words or bigrams that are blacklisted are displayed in the manner in Figure 6.4, where the two blacklisted words ”opened” and ”error” have been found. For each hour, the count of the blacklisted word or bigram is listed. If nothing blacklisted is found



Blacklist

Words

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| opened | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| error | 5 | 4 | 4 | 16 | | 17 | 16 | 16 | 13 | 16 | 17 | 16 | 16 | 33 | 19 | 16 | 16 | 16 | 8 | 12 | 16 | 17 | 4 | 4 |

Bigrams

Nothing blacklisted today.

**Figure 6.4:** The visualization of occurring blacklisted words and bigrams

during the selected day and log file, the phrase "Nothing blacklisted today." is shown.

In order for the user to gain an insight into the contents of the log files, the most frequent words and bigrams are shown in a *word cloud*, as is done for bigrams in Figure 6.5. To the left, the cloud containing information from the selected day is seen, while the rightmost cloud contains data pertaining to the normal state of the log file. There is a pair of word clouds for each hour, and the displayed hour is selected by buttons along the top. If there is an anomaly during an hour, the button is colored yellow to indicate this. The size of a word or bigram in the cloud indicates how frequently it occurs in the selected log file during a specific hour. The size is proportional to the square root of the actual count. There is an identical view that instead displays words.

The different groups of messages are listed per log file as can be seen in Figure 6.6. Only message groups that display anomalous behavior are listed, and they are grouped by hour. To the left of the message itself is an indicator of how the message is anomalous. The green dash indicates the range of the normal number of messages, while the red bar represents today's number of messages and either stretches too far or not far enough to end within the dash. A tooltip shows the exact normal and the actual values. The messages have been blurred to protect the confidentiality of 3bits' data.

## 6.2 The usefulness of the prototype

This section describes how users experience the prototype, based on two interviews with members of 3bits Consulting AB. Main points of interest are if the users consider the prototype useful and if it assists in handling log files.

Both interviewed individuals expressed that the prototype would be useful when handling log files. One individual stated that the prototype was able to provide a good overview of the system, as a lot of information was shown without overwhelming the user. Both users could give concrete examples of past incidents where the prototype could have assisted them in finding, or preventing, an error.

The only feature that the users were doubtful about was the inclusion of bigrams. One user stated that there may be situations where they might be helpful in understanding

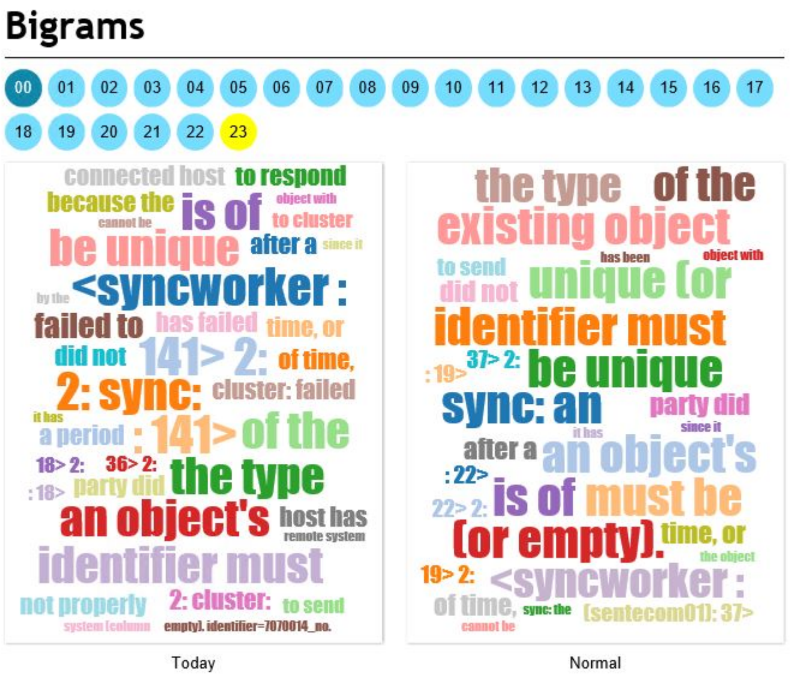

Bigrams
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
18 19 20 21 22 23
connected host to respond because the is of object with cannot be to cluster be unique after a since it by the <syncworker : failed to has failed time, or did not 141> 2: of time, 2: sync: cluster: failed it has a period : 141> of the 18> 2: 36> 2: the type :18> party did an object's host has remote system identifier must not properly 2: cluster: to send system (column empty). identifier=7070014_no.
the type of the existing object has been object with to send did not unique (or identifier must : 19> 37> 2: be unique sync: an party did it has since it after a an object's : 22> 22> 2: is of must be (or empty). time, or the object 19> 2: <syncworker : of time, sync: the [sentecom01]: 37> cannot be
Today
Normal

**Figure 6.5:** The visualization of the frequency of different bigrams in a log file

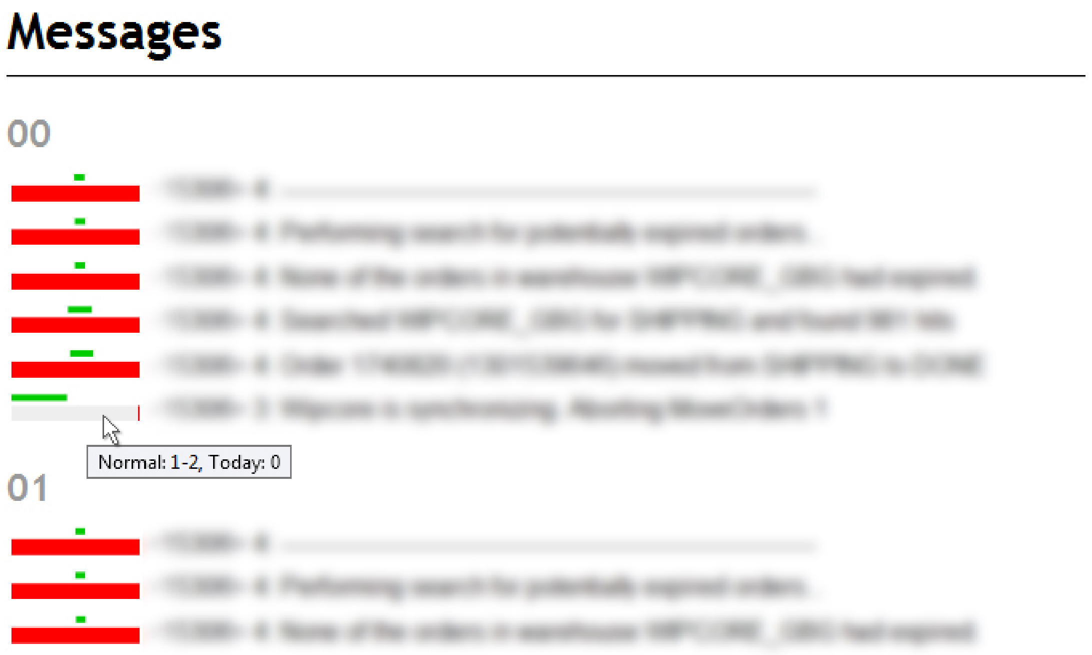


**Figure 6.6:** The visualization of different groups of messages. The messages are blurred to protect the confidentiality of 3bits' data.

the contents of the log file, but that they did not view bigrams as useful. There were also an opinion that there were an overlap between words and bigrams in terms of information and functionality.

The users had some suggestions regarding functionality that they felt was missing. The majority were minor suggestions pertaining to the visualization, and can be found in Section 6.3. One suggestion was to have a dynamic blacklist that could be updated and reflected in all visualization at any time, similar to a search function that lists the occurrences and normal count for any desired word. Another suggestion was to have a file that explains how the prototype processes and calculates different limits, as a guide for a user who wants to understand more about the prototype.

## 6.3 The effectiveness of visualization

This section describes the effectiveness of the visualization found in Section 6.1 as based on two interviews with members of 3bits Consulting AB. The interviews revealed that the visualization worked well but could use some adjustments. The users had almost no issues in navigating the prototype, except for how to use the double arrows for changing the date by a week. They expressed that they found the interface to look good and contain a lot of information without being cluttered. Anomalies were spotted at once.

One point of confusion was what the different parts of the graph over total counts represented, as seen in Figure 6.2. Both users were able to accurately guess what the black line, dark green line, and green area represented, but they expressed that a legend over the meanings would help make the interface clearer. The representation of the anomalies was very clear and caused no bewilderment, and the weekly view was thought to be especially useful.

The blacklist posed no issue, as the users at once understood that it contained undesired words. One user expressed that the counts per hour for each word were useful. As mentioned, one user expressed a desire to be able to update the blacklist and have the change propagate backwards in the analyzed files.

The visualization of words and bigrams was thought to be "good looking", and the size of the words were easily understood to represent their frequency. As mentioned in Section 6.2, the only part of the visualization that was not found to be useful was the visualization of the bigrams.

The listing of abnormal message groups, as seen in Figure 6.6, was viewed as the least clear section. Neither user at first understood the contents of the section when viewing a file with a single abnormal message group per hour, but it became clearer when there were more than one message group for each hour. The section was thought to benefit from being labeled "Anomalous message groups" rather than just "Messages", and one user expressed a desire to label the hours more clearly.

It is interesting to note that the users most often found anomalies by looking at the total counts of the number of output lines, and then scrolled down to view more details about the anomalous time period. The total counts of words and bigrams were rarely used, and the blacklist, words, bigrams and message groups were used to gain more information about the anomaly rather than discovering it.

The main feature lacking in the visualization according to the users was a total overview of the system, by for example indicating in the list of log files if a certain file required attention. The users felt that investigating all log files just to see if an error had occurred was not user-friendly enough, and would require too much time. The order of the list of log files was also experienced as somewhat messy. The users suggested showing the entire folder structure or being able to sort the list as possible improvements. Finally, the buttons for changing the date would benefit from having tooltip, as the users did not understand clearly that the double arrow changed the date by seven days.

## 6.4 Time to process and storage used

This section will describe how the prototype used the resources of time and storage space. This is done for each of the three datasets. For comparison, the original size of the log files can be found in Table 6.1. Note that there are two rows concerning 3bits, where the first lists the size of all input files and the second lists the size of a reduced set of the log files. The difference between these rows will be discussed later in this section.

**Table 6.1:** Size of log files used

| Source | Data size (GB) |
|---|---|
| 3bits (all files) | 314 |
| 3bits (modified) | 76 |
| Ubuntu 14.04 | 0.014 |
| Windows 8.1 | 0.009 |

The time to process a single day of all log files provided by 3bits Consulting AB can be found in column four of Table 6.2. The table lists the duration for the different steps, along with the size of data output in each step. The second column includes all the files in the period when accounting for the size, while the duration of processing in the third column was found by processing three days of new data. Due to the amount of stored log files, it was not feasible to process all of them at once to find the total processing time. The average total time to process a single day is 1 hour and 34 minutes. The total storage needed for all files in the period was 147 GB, which is 167 GB less than the original log files.

Due to an issue where the processing machine ran out of memory when attempting to group messages, words and bigrams for the two largest file, these preprocessing steps were skipped for those two files specifically. As this affects the processing time and storage for all stages, the results when excluding these two files can be seen in Table 6.3. The modified input size of the log files were 76 GB. The average total time to process a single day was 1 hour and 12 minutes. The storage used was 19 GB, which is 57 GB less than the original log files.

**Table 6.2:** Time to process and storage used for data from 3bits Consulting AB data during February to May 2015

| Phase | Output data (GB) for entire period | Processing time for three days | Average time per day |
|---|---|---|---|
| Preprocessing | 94 | 4h 35min 35s | 1h 31min 52s |
| Analysis | 46 | 7min 21s | 2min 27s |
| Anomaly detection | 7 | 3s | 1s |
| Visualization | <1 | <1s | <1s |
| Total | 147 | 4h 42min 59s | 1h 34min 20s |

**Table 6.3:** Time to process and storage used for modified set of data from 3bits Consulting AB data during February to May 2015.

| Phase | Output data (GB) for entire period | Processing time for three days | Average time per day |
|---|---|---|---|
| Preprocessing | 9 | 3h 4min 7s | 1h 1min 22s |
| Analysis | 4 | 7min 20s | 2min 27s |
| Anomaly detection | 6 | 3s | 1s |
| Visualization | <1 | <1s | <1s |
| Total | 19 | 3h 4min 7s | 1h 11min 30s |

A comparison between the sizes of the two different approaches to 3bits' data can be seen in Figure 6.7. The first and third bars display the size of log files for the entire dataset and the reduced dataset respectively, while the second and fourth bars display the size of the corresponding processed files. Note that visualization is not visible in the bars due to its small size.

A summary of the processing time and storage used when processing log files from Ubuntu is available in Table 6.4. Note that the storage is listed in MB rather than GB. The third column lists the time to process log files from the entire period, 30 days, while the fourth column lists the average processing time for a single day. As can be seen, processing one day takes 4 seconds. The storage space used for the processed data for all the log files is 61 MB, which is four times more than the 14 MB of input data.

In Table 6.5, processing time and storage used are listed for the processing of Windows log files. Note that storage is once again listed in MB. The third column lists the time to process 227 days, while the fourth lists the average processing time for one day. The average processing time for a single day is approximately 1 second. There is 38 MB

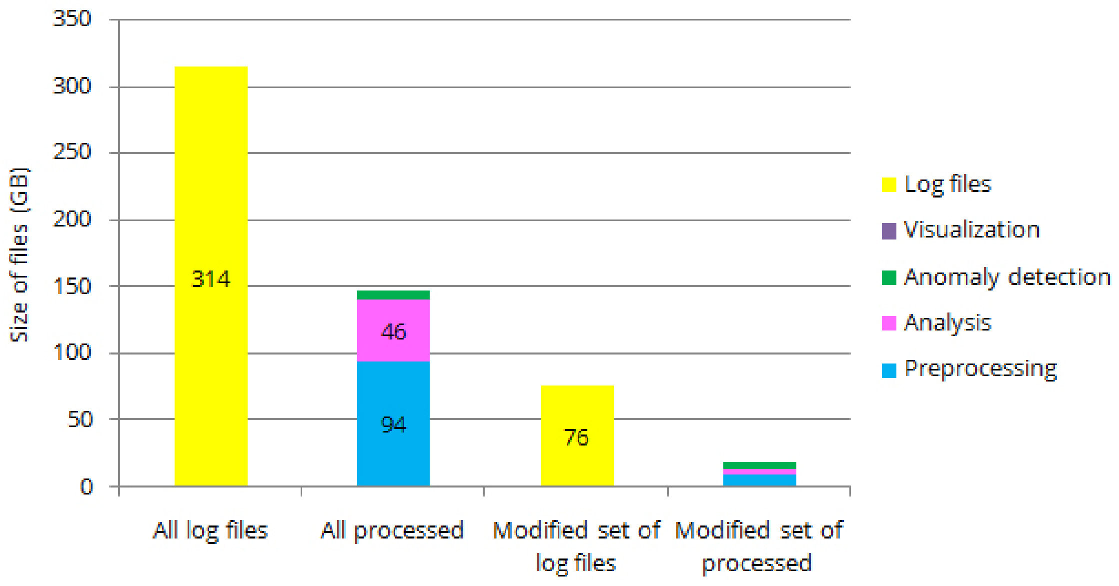


**Figure 6.7:** The size of log files and processed files related to 3bits Consulting AB

**Table 6.4:** Time to process and storage used for Ubuntu data from 2015-01-05 to 2015-04-29, containing files from 30 days

| Phase | Output data (MB) | Processing time | Average time per day |
|---|---|---|---|
| Preprocessing | 39 | 1min 24s | 3s |
| Analysis | 19 | 29s | <1s |
| Anomaly detection | 3 | 1s | <1s |
| Visualization | <1 | 1s | <1s |
| Total | 61 | 1min 55s | 4s |

of output data, which is four times more than the input 9 MB of data.

The sizes of files related to the Windows dataset as well as the Ubuntu dataset is found in Figure 6.8. The first and third bars display the size of log files for the entire datasets, and the second and fourth bars display the size of the corresponding processed files. Note that visualization is not visible in the graph due to its small size.

The time to process log files could not be modeled since it depended on factors that are difficult to quantify. Specifically, the contents of the files affect the processing time, and somewhat the storage used. An example of this is log files containing a large number of different messages, as they required more time and storage than files with a single message group.

**Table 6.5:** Time to process and storage used for Windows data from 2014-08-05 to 2015-04-28, containing files from 227 days

| Phase | Output data (MB) | Processing time | Average time per day |
|---|---|---|---|
| Preprocessing | 21 | 2min 46s | <1s |
| Analysis | 13 | 40s | <1s |
| Anomaly detection | 4 | 35s | <1s |
| Visualization | <1 | 3s | <1s |
| Total | 38 | 4min 4s | 1s |

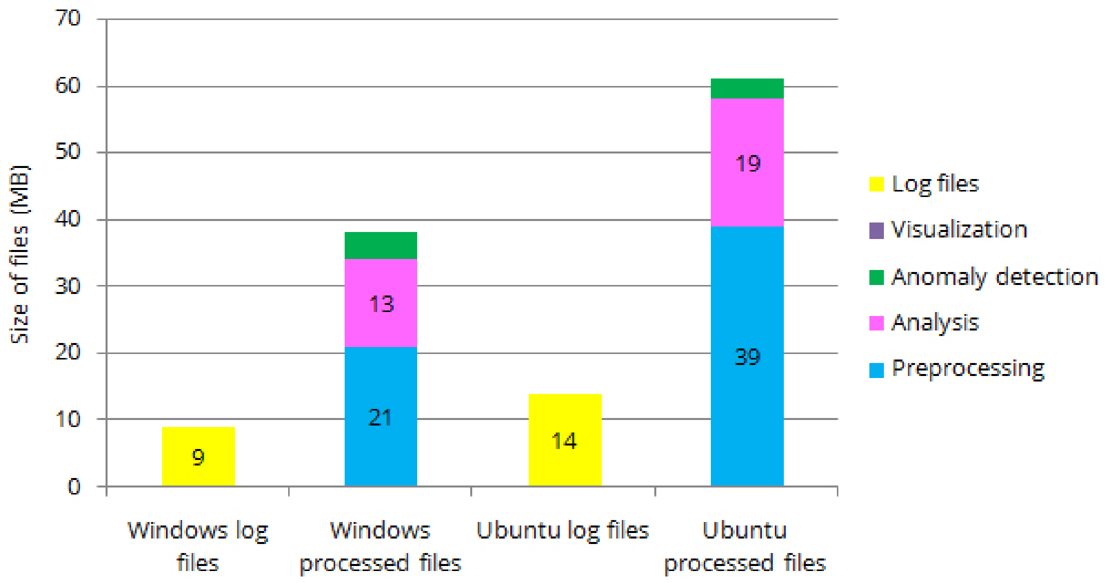


**Figure 6.8:** The size of files related to Windows 8.1 and Ubuntu 14.04

## 6.5 The effectiveness of the anomaly detection

This section discusses how effective and accurate the anomaly detection is. For readability the 3bits, and Windows and Ubuntu datasets are discussed separately.

### 6.5.1 Files from 3bits Consulting AB

A general result found was that the anomaly detection regarding words and bigrams was not useful, as the exact information regarding which word and the exact values was not included in the visualization. Though the information could be accessed by opening a certain file, and contained useful information about the shift in content, it could not be utilized by the user to find anomalies easily. The anomaly detection on message groups

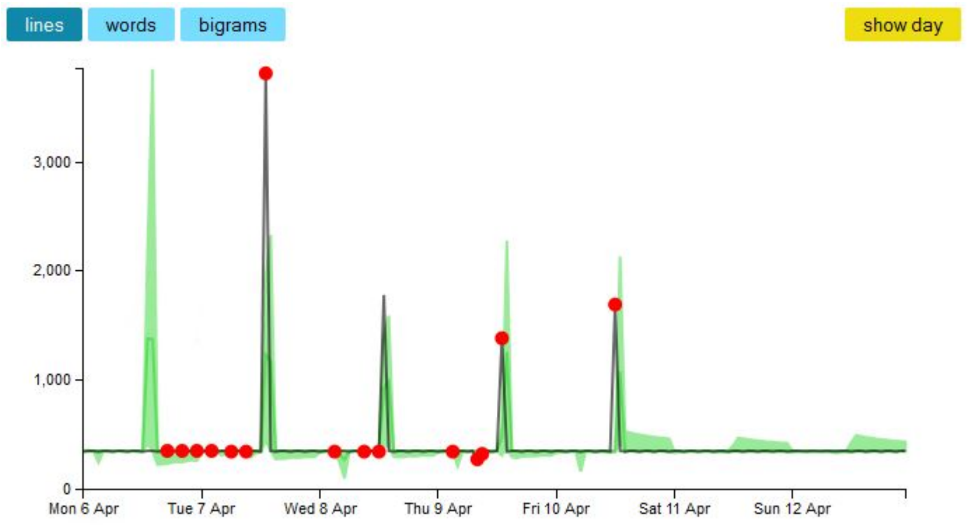


**Figure 6.9:** The number of lines from File A containing spikes during a week, while displaying a number of incorrect anomalies

was useful to determine how a log file had changed once an anomaly had been raised, as it spotted messages that varied in an abnormal way.

To describe more specific results, the log files from 3bits are divided into a small number of groups based on their behavior. For each group, typical points of interest will be described. This includes raised alarms, false alarms, and missed anomalies. The discussion will mainly concern the total count of output lines, as total counts of words and bigrams closely follow the same patterns. The log file names are anonymized in order to not disclose their contents.

The first type of log files are such that they consist of a constant output, henceforth called *baseline*, which is interjected with spikes in output at regular intervals. The example of such a file, File A, outputs approximately 2,000 lines every weekday at 13:00, and approximately 400 lines at all other times. In Figure 6.9, the number of lines per hour is displayed over a week. Usually, the spike at 13:00 is extra large on Mondays to compensate for the weekend which only outputs the baseline, but this week only the baseline was output on Monday. To compensate, the spike during the Tuesday was 4,000 lines rather than the normal 1,000 lines, and was marked as an anomaly. This is correct due to the number of lines normally being smaller on Tuesdays. However, the absence during the Monday was not noted as an abnormality, which is incorrect. As can be seen during the Thursday and Friday, the spikes are incorrectly marked as anomalous. This is due to the output occurring at 14:00 rather than 13:00 the previous week. Finally, note that a number of values along the baseline are marked as anomalous. This is incorrect, as their deviation in all but one case is negligible.

Figure 6.10 also shows File A, but from a later week. The Figure displays correct behavior when marking that it is abnormal to output 6,000 lines rather than the normal 3,000 lines during the Monday, as well as the decreases in the baseline. Note that, as in

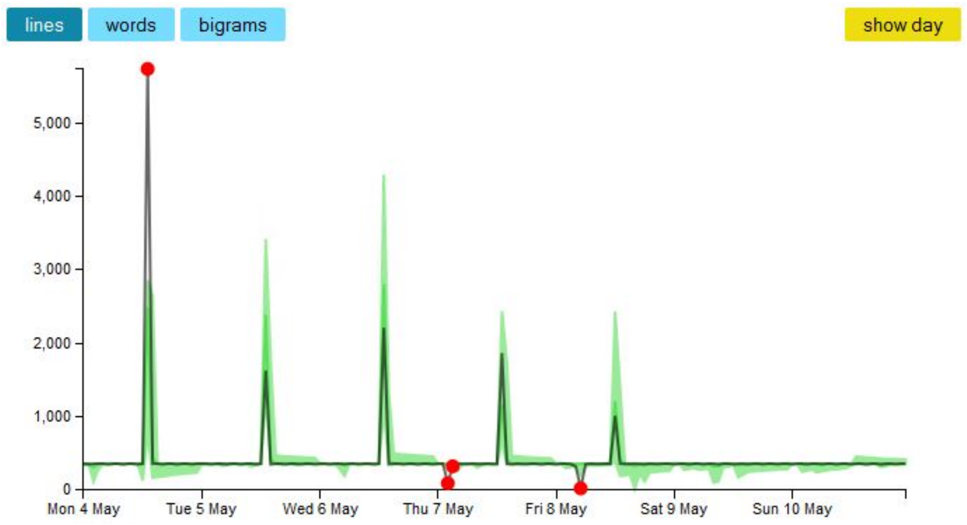


**Figure 6.10:** The number of lines from File A containing spikes during a week, while displaying correct behavior

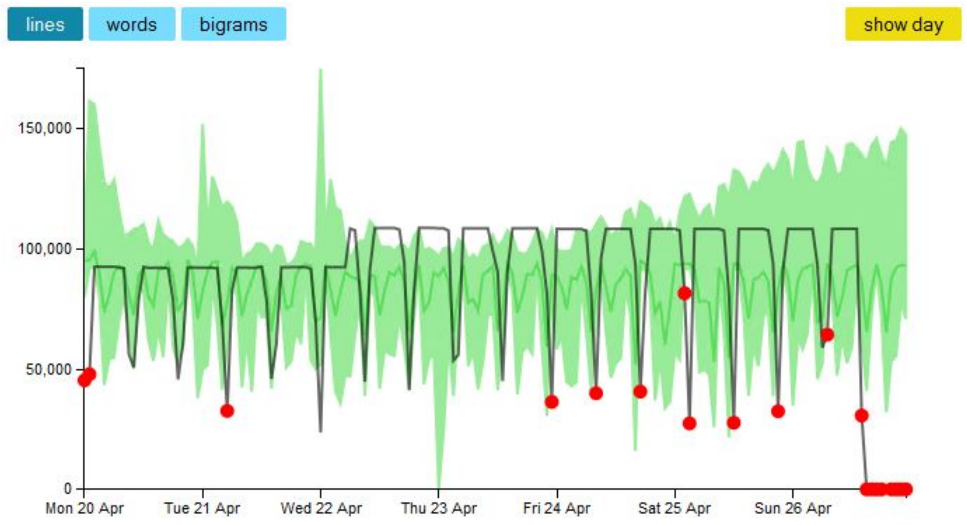


**Figure 6.11:** The number of lines from File B containing plateaus during a week

Figure 6.9, the weekend is correctly not marked as being abnormal.

The second type of log files contain a number of *plateaus* and dips in output, and can almost be considered the inverse of the first type. File B is such a file, it constantly outputs around 100,000 lines, but reduces the output to around 50,000 lines every nine hours. The number of lines for File B over a week can be seen in Figure 6.11. Note the shift in the normal baseline during Wednesday. As can be seen, a lot of the dips are marked as anomalous, due to occurring at nine hour intervals rather than at fixed times. During the Sunday, the absence of output log lines is correctly marked as anomalous.

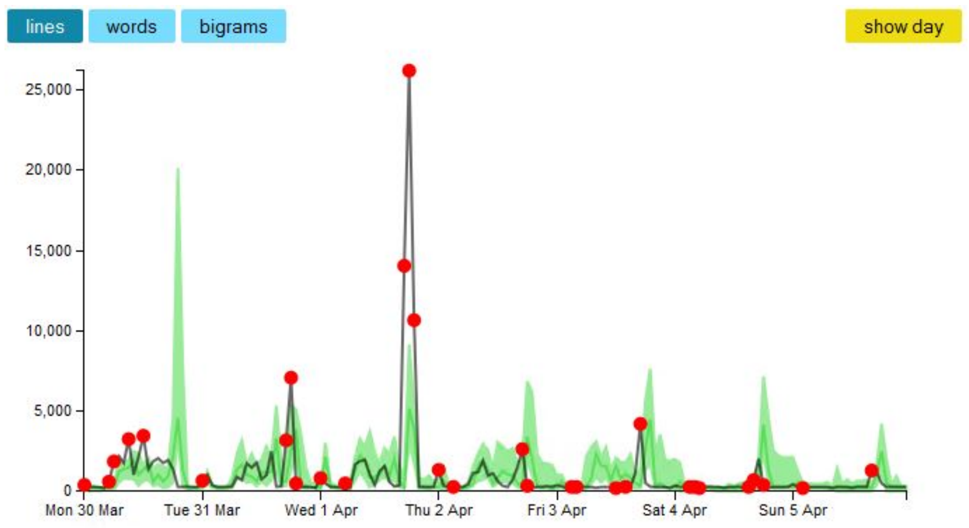


**Figure 6.12:** The number of lines from irregular File C during a week

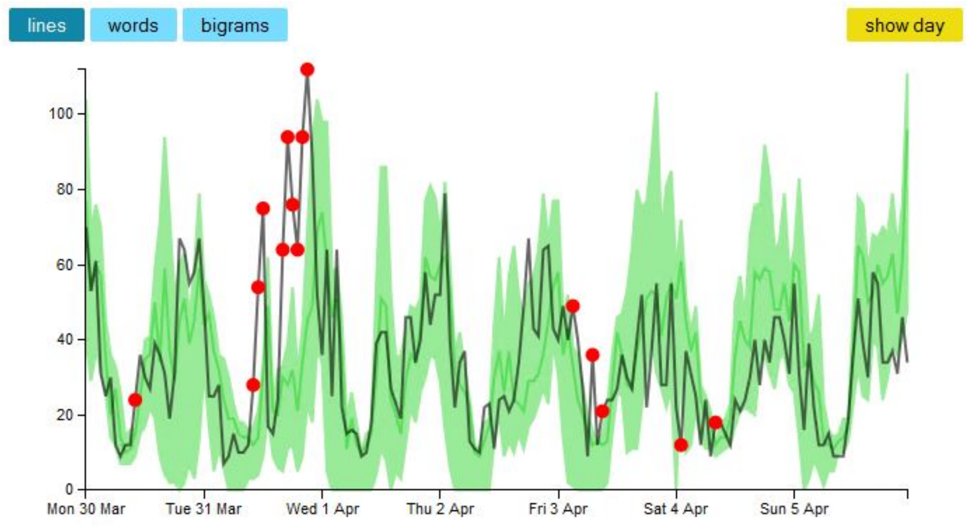


**Figure 6.13:** The number of lines from irregular File D during a week

The third type of log files are constantly fluctuating. They can contain an overall pattern, but the number of output lines varies from hour to hour. An example of this is File C, which normally outputs somewhere between 0 and 5,000 lines each hour. As can be seen in Figure 6.12, a lot of anomalies are found. Except for the anomalies shown during the Wednesday, it is likely that most anomalies are superfluous.

Another example of a fluctuating file is File D. As can be seen in Figure 6.13, it displays a more regular behavior than File C. The anomalies marked during the Tuesday are correct, while the anomalies for Monday, Friday and Saturday are not.

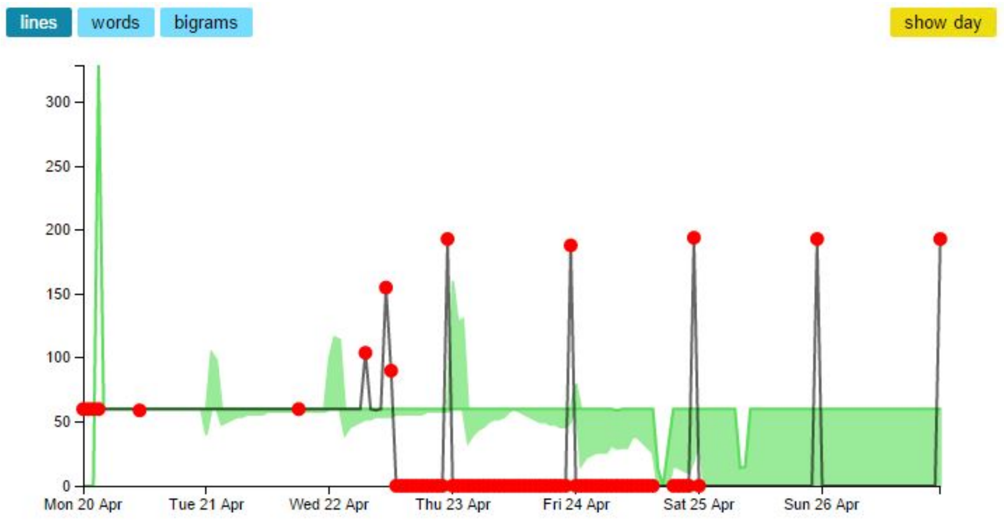


**Figure 6.14:** The number of lines from File E during a week, showing a shift in normality

### Accounting for a shift in normality

The prototype is able to account for a shift in normality well. An example of this can be seen in Figure 6.14, which shows File E and how the baseline for the number of rows changed on Wednesday. After three days, on the Saturday, the new baseline has been accepted. The spikes in the number of lines are not accepted until a week after the baseline is.

Another example can be seen in Figure 6.15 (note that the figure is over a period of 20 days). It shows File F, which increased from around 2,000 lines per hour to around 100,000-500,000 lines per hour. This is first correctly seen as anomalous, but as it continues to increase the assumption is made that the behavior is correct. The two anomalies in the middle of the green area are correct, but due to the large number of days displayed in the figure, the dip in maximum value cannot be seen in the picture.

To illustrate the effect of skewing, Figure 6.16 can be used. It shows the same file and duration as Figure 6.15, but no skewing has been done. Thus, the maximum value calculated for next week is purely based on the numbers from last week and the current date. This means that the maximum value is always a week behind and, in this case, the normal state does not adapt to the change until after two weeks when the standard deviation has grown enough to account for what the max value still misses. This is starkly contrasted to Figure 6.15, where the threshold values are skewed with respect to the changes in the last three days. In barely a week, the maximum threshold adapts to the continuous change. It then continues to grow for a while, allowing for more change, and finally settles down as the curve stabilizes. As the average line is not affected by the skewing, the user can still see the expanding difference between the current day's values and the values that are considered normal.

As can be seen in the previous examples, the false alarms are numerous. They make

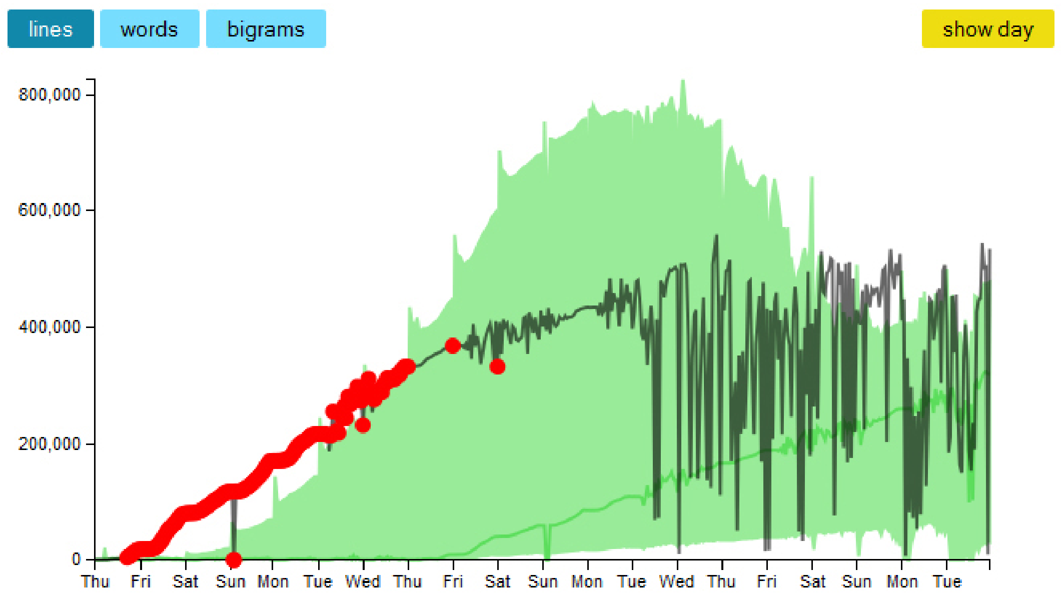


**Figure 6.15:** The number of lines from File F during 20 days, showing the adaptation when skewing with weekly change

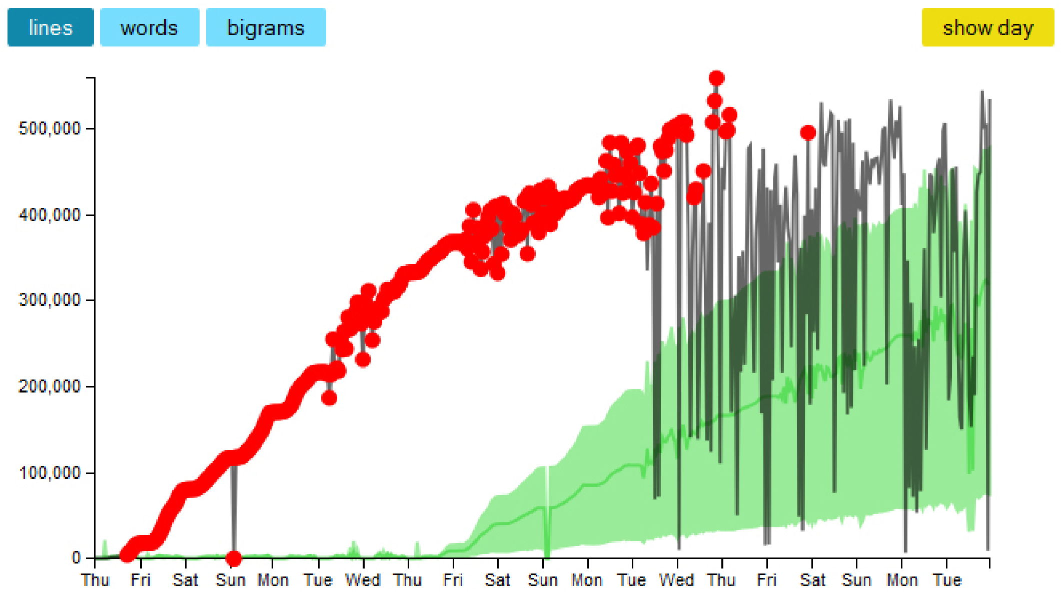


**Figure 6.16:** The number of lines from File F during 20 days, showing the adaptation with no skewing

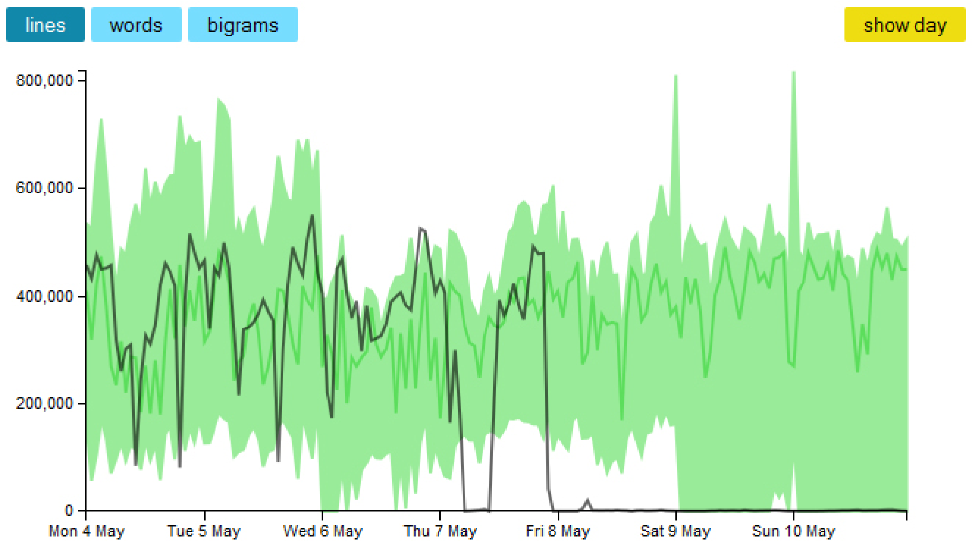


**Figure 6.17:** The number of lines during a week for File G, with a reduction in output during the Friday

up a very large part of the incorrectly handled data points, and can thus be considered one of the biggest problems of the prototype along with the inability to process very large files.

### Finding known errors

To evaluate the accuracy of the anomaly detection, a list of 14 known errors was used. The errors had a variety of side effects on the log files, ranging from error messages to increased output to no output. Out of the errors, eleven were found and correctly marked by the anomaly detection. The three missed faults were caused by three different reasons. The first miss was due to a premature adaption to an erroneous state. The second miss regarded File G that had its output reduced to zero, as seen in Figure 6.17, and it was not caught due to the high variability in the file. The third missed error was of a too complex nature to be noticed by the prototype, involving the site of a specific country not being available. This was not reflected in the total or individual counts of words since the country was not prominent enough during normal operations.

## 6.5.2 Files from a Windows and a Ubuntu machine

Both the Windows 8.1 and the Ubuntu 14.04 datasets are rather sparse, as they are taken from machines that are not always in use. The occurring events were very irregular which affects the analysis, and virtually all events were marked as anomalous. An example for this from the Ubuntu auth log file can be seen in Figure 6.18, while Figure 6.19 displays a Windows program log file. Due to days where no log files were produced, not even empty ones, the analysis was reset quite often. This is especially true for the Ubuntu

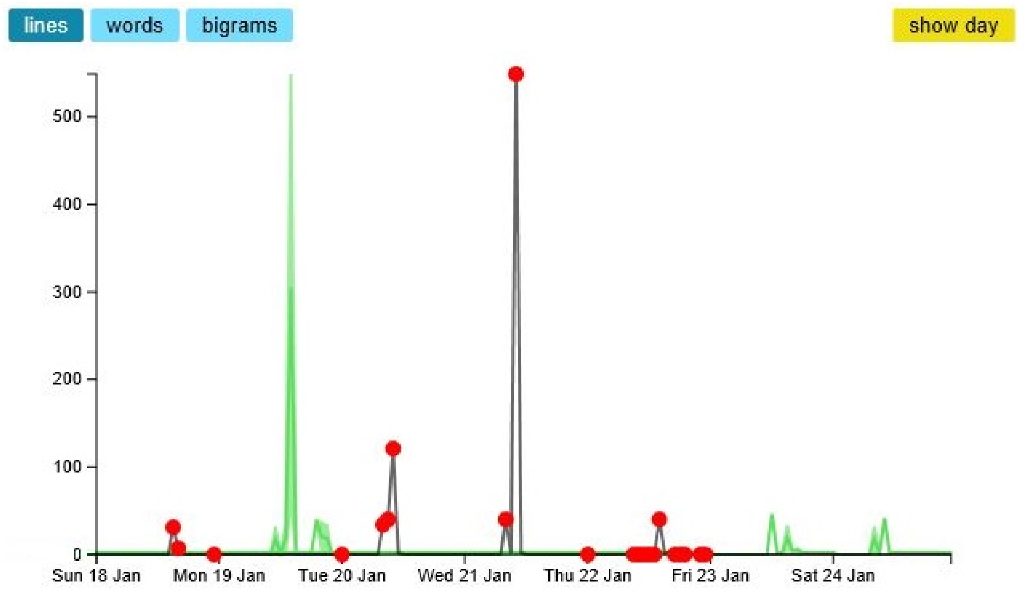


**Figure 6.18:** The number of lines per hour during a week for an Ubuntu auth log file

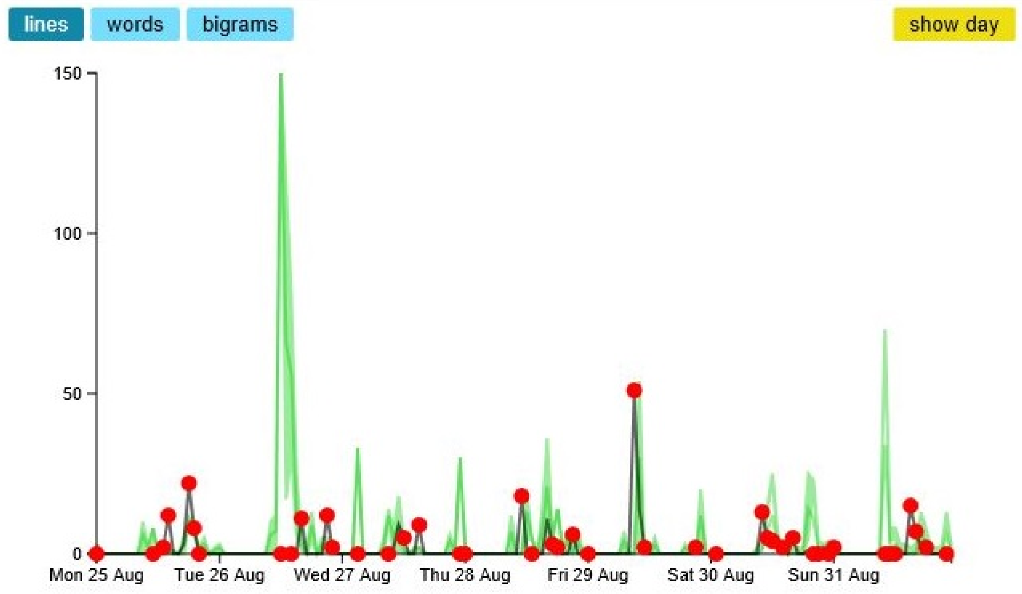


**Figure 6.19:** The number of lines per hour during one week for a Windows program log file

log files. The Windows log files were more regular, and were thus able to, through the analysis, accept more values, as can be seen in Figure 6.20.

The grouped log lines worked very well for both Windows and Ubuntu log files, in

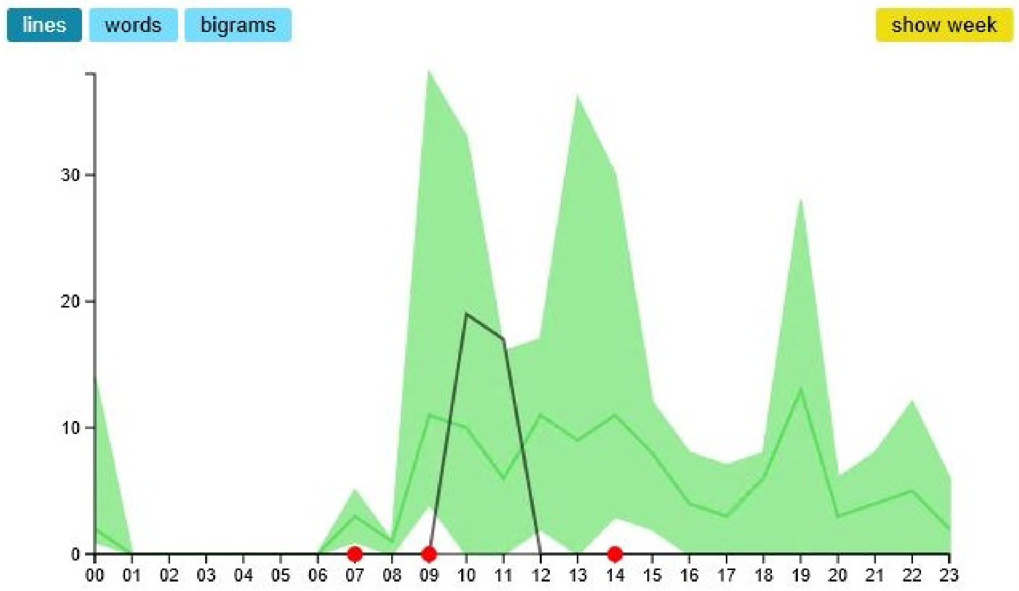


**Figure 6.20:** The number of lines per hour during one day for a Windows program log file

part due to the low number of different messages in the files. They provided important insights into the files and were more useful than words and bigrams in understanding the changes of the log files. Due to the irregularities a number of messages were incorrectly listed as anomalous.

An issue caused by days that lacked log files was that the analysis was not performed during days without files. This led to an old state of normality being used for longer than prudent.

### 6.5.3 Grouping of messages

Sometimes during the preprocessing when grouping messages, the machine handling the processing (equipped with 4 GB RAM) ran out of memory. This occurred for two different log files whose sizes were close to 2 GB each when they contained a large number of different messages. Files that lead to this error were not grouped by message, but other metrics such as the total counts and words were still found. Unfortunately, the memory did not suffice during the analysis of words and bigram either, specifically during the merging of tables. Thus, the analysis of these two files was restricted to the total counts.

One issue with using the longest substring to group messages was found when a log file consisted of long lines with a lot of variables. For example, the lines in Listing 6.2 are grouped separately though it is obvious to the human eye that they follow the same pattern. This is due to the large number of variables placed at intervals in the line, as they cause the longest common substring to become short. Not only did this lead to poor grouping, it also lead to long processing times due to the large number of possible groups that any new message could be sorted into. This lead to files with a large number of messages taking a long time to process.

**Listing 6.2:** Two log lines not grouped together. Modified for brevity.

```
New stock for product '3158701': Stock = 187, StockLow = 159,...
New stock for product '3207617': Stock = 691, StockLow = 552,...
```

One case of severely incorrectly grouped messages was found in the final result. This was due to a rather long common preamble and short actual content. This faulty grouping lead to some confusion regarding the change of a file.

## 6.6 Flexibility of data format

As mentioned in Section 3.1.3, the main difference between 3bits' log files and the private log files was the formatting of date and time. By utilizing the settings file, the program was able to handle the different formats without any problem.

Though the contents of the private log files were very different from the contents of the e-commerce log files, including being written in different languages, no issues were encountered during the preprocessing or analysis phase. The message grouping performed well regardless of the contents of the row, and non-English characters did not pose a problem.

# 7 Discussion

This chapter will discuss how the prototype conforms to the initial goals of this thesis. Focus will be placed on the results in relation to the goals, but topics with a more general scope are also covered.

## 7.1 Visualization

The visualization investigated a few different ways of showing information, from a line graph to a word cloud or just listing anomalous data as for message groups. These can be seen in Section 6.1, and the users' experiences are described in Section 6.3.

The overall reception of the interface was very positive, since the users liked the visual style and were able to navigate it well. A couple of minor points such as clarifying some titles and adding tooltips should be done, but the only change that needs to be done to the interface itself is the addition of an indication of whether a file should be viewed or not. This could for example be tied to the number of anomalies in the file, effectively giving the user an overview of the system. In fact, this was thought of very early in the design process, but was never realized in the prototype.

There is a difference between displaying information purely to find faults, and displaying it to give the user an understanding of the system. The word clouds were mostly informative as there was not an exact, clean way to include information about exactly which words were anomalous. The prototype calculates such data, but it is currently not displayed, and it is possible that displaying it would cause information overload. The visualization of bigrams overlaps in functionality with the visualization of the words, and it is possible that they should not be shown at all but rather used to calculate changes in a file. While word clouds are good tools for showing which words are more common in a dataset, comparing two word clouds is rather straining for the user. Since the words are positioned randomly, finding a pair is rarely done in an instant. A solution to this could be to use the same color for identical words and to highlight the word in both clouds when a user hovers over a word.

While the purpose of the word clouds was to give interested users more insight into the contents of the log files, it was believed that users would benefit more from the message groups focusing on anomalies rather than displaying every single message in correlation to each other. The message section was the most experimental and least understandable part of the visualization. The goal was to show the user the range of values that was considered normal and to compare that to the anomalous value, and in doing so show just how abnormal the value was. One first idea of how to do this was to display a table resembling a stock market table. Such a table would include the message, the anomalous value, the average value, perhaps the standard deviation, and the value and percentage with which the anomalous value diverged from the normal range. This representation was however believed to contain too many figures if the number of anomalies was high, and so several other ways to visualize the difference were explored. As the charts used in the prototype were scaled to have the same width, it was impossible to tell apart a chart with normal value 0-1 and abnormal value 3 from a chart with normal value 0-1,000 and abnormal value 3,000. This meant that the user had to hover over the chart to find out just how alarming an anomaly was. In retrospect, a table would probably have been a better idea as it would give the users all the information without requiring them to make any extra effort.

Overall, there are a number of minor adjustments that can be done to the visualization to make it more elegant and clearer. However, the current interface is understandable and provides an easy overview, and as such the prototype can be used in its current state.

## 7.2 The usefulness of the prototype

As stated in Section 6.2, all interviewed users stated that the prototype would be useful to assist them in their daily work. The fact that the users were able to name specific incidents where the prototype could be used indicates that it would be able to assist in finding errors. Feedback from the users also indicates that the prototype may be able to shift the log handling into a more proactive approach, where the log files are continuously monitored, leading to a prevention of error or at least reduced error handling time.

The doubtfulness regarding the usability of bigrams indicates that it may not be interesting to include this part in the visualization. As mentioned in Section 6.3, during the test it was noticed that the users also almost completely disregarded the graphs for the total number of words and bigram, so these three metrics could instead be used to calculate the changing nature of the contents of the log files, or be used as a basis for anomaly detection. The main point is that the information is useful, but not very interesting to the user in the current form of visualization.

Regarding one user's wish of enabling dynamic updates of the blacklist, being able to find the exact numbers for the occurrence of certain words would undoubtedly be useful to find the cause of an error. However, when considering that the intended use of the blacklist was to inspect a suspect word to find its change, this functionality would fit better into a separate search function. The blacklist should be used only to flag words that are always considered dangerous or incorrect.

To satisfy the desire for deeper explanation into how the prototype works, a tutorial could be considered. Not only would this familiarize new users with the interface, it could also be used to explain less obvious points about the data processing. However, the users should not need to understand exactly how for example limits are calculated, as long as it is clear how they should be utilized by the user.

## 7.3 Time to process data and storage used

The average time to process the modified set of files from 3bits Consulting AB was 1 hour and 12 minutes for one day, which indicates that the prototype is fast enough to be able to compile hourly summaries and anomaly detection. The bulk of the time was spent on preprocessing files as can be seen in Table 6.3, specifically on working with individual words and bigrams. If processing time is to be reduced, preprocessing is where the most is to be gained.

The time to process log files from the Ubuntu and Windows machines was very short, although the storage used was far more than the size of the actual log files. This indicates that small log files are easy to process, but they may not benefit as much from it. Though the added visualization was helpful, it was feasible to view the actual log files.

Results for the storage used are acceptable. Though the data from 3bits with partly processed files excluded shows a 75% decrease in the storage used, while the complete set is 47% smaller. The excluded files from 3bits made up 86 GB (91%) of the preprocessed data, and 42 GB (91%) of the analyzed data. They were 238 GB (76%) of the total input, which suggests that there is not a linear relationship between input size and the size of the analyzed files.

The processed data was four times larger than the input data for both the Windows and Ubuntu dataset. This is not acceptable, though the size difference was less than one gigabyte. Due to the small size of the Windows and Ubuntu dataset, the size of files created during preprocessing and analysis surpassed the size of the input data. For this prototype to be useful for systems that do not output a large number of log files the processing would have to be more efficient in terms of space, which may not be possible due to the minimum number of files needed to analyze all different aspects.

Due to the size of the two largest files, the number of words and bigrams are very large. As several files list different attributes for the words and bigrams, this leads to significantly larger files. The relationship between the number of words and the number of possible bigrams is a combination and thus grows very quickly, and it was found that the number of occurring bigrams quickly becomes larger than the number of occurring words. Out of the 86 GB of preprocessed data, 52 GB (60%) consisted only of bigram data. The time and space needed to store and process bigrams is too large, and must be reduced during future iterations of this prototype. It is even possible that bigrams are not useful enough to justify the storage space and processing time needed.

To reduce storage and processing time when handling word, bigram, and message groups, a limit to the length of the words could be used. A word longer than the limit would only have its beginning up to the limit considered, for example the first 100 characters. Depending on the limit, this could reduce the size of word and bigram files, as well as making the comparisons faster. The usefulness for word and bigram files would be limited, as there are few words that are truly long, but this could possibly lead to the grouping of messages taking significantly less time. However, it is important to keep in mind that limiting the length also increased the possibility of two dissimilar messages being grouped together.

As mentioned in Section 5.2, words and bigrams that only occur once are not included in the analysis. This limits the size of the word and bigram files, which lowers processing time, but it may not be reasonable to assume that words and bigrams that only occur once are irrelevant. Though this thesis assumes that words are important if they occur often, a single occurrence of “error” may be more important than the normal behavior of outputting “sleeping” a thousand times. To further increase the usability of this prototype, rarely occurring words should also be taken into account, though this involves handling ever-changing values of variables as well.

The choice of storing all files on a tab-separated values format was made to facilitate the visualization as the D3 library already had built-in support for reading such files. While this was good for visualization, it had a rather negative effect on the size of the files. If visualization had been better integrated with the rest of the system, files could have been stored in a more compressed manner. Some files could probably be stored on another format even without replacing the method of visualization. As discussed in Section 7.1, bigrams could eventually be removed from visualization and only used in the analysis of files. This means that the bigram files, which currently make up 9 GB (44%) of the processed files from the 3bits modified dataset, would only need to be readable by the program and thus their size could be heavily reduced.

The current method of finding the normal state of words and bigrams involves merging two tables. This is a costly operation due to such tables often being very large, and could possibly lead to using too much memory during the processing. Looking up values for a specific word or bigram, which is done quite often, is also very demanding if the file is large. It is possible that using tables is not well suited to store and compare such large amounts of data.

The action of grouping messages is rather taxing. As described in Section 6.5, one particular file resulted in poor groups and an unreasonable processing time. As this was specifically due to the usage of longest common substring to compare words, it may be worth considering some other method for finding if strings are roughly similar. However, as can be seen in Chapter 4 and 5, a number of different methods were tried and found to all have negative aspects.

## 7.4 The effectiveness of the anomaly detection

The anomaly detection of log files from 3bits performed fairly well, but raised too many alarms. Consistent data with a small variability was handled best, while irregular data and data where the pattern was not based on the time of day were handled less well.

The problems found in Section 6.5 were caused by a few different factors. The false alarms that occur during the baseline hours in Figure 6.9 are due to the usual exactness in the file. This leads to a very small standard deviation and consistent maximum and minimum values, and thus even a small deviation will be marked as anomalous. While it is true that such deviations are anomalous, they are most likely not caused by an error and are thus uninteresting to the user. It would be an improvement to ensure that very small deviations from the baseline are not marked as anomalous.

In quite the opposite manner, the failure to mark the missing output during the Monday as anomalous is due to the large variability in the output. As can be seen in Figure 6.9, the average is approximately 1,500 lines during the peak hour, while the historical maximum is 4,000 lines. This leads to a large standard deviation that allows the output to be reduced to the baseline without causing an alarm. An identical issue can also be seen in Figure 6.17. The issue of using standard deviation to find anomalous values is mentioned in Section 4.1.1. Though the anomaly detection worked well, issues such as these raise the question of whether it is reasonable to assume a normal distribution for the total counts of log lines, words and bigrams.

Another problem was due to the inherent assumption of the prototype that logs are produced according to a pattern related to the time of day. This was not true for Figure 6.11, where the logs followed a nine hour cycle, or for Figure 6.12 which had no clear pattern at all. Figure 6.13 is another example that has a rough pattern with too much variability to be handled well. If a log file lacks a periodicity, it will raise many anomalies due to the fact that the prototype is not able to learn the normal state of such files. This is naturally a weakness, but adapting to irregular data may cause the anomaly detection to become less reliable for regular data.

As mentioned in Sections 6.3 and 7.2, the visualization of words, bigrams, and message groups were used to investigate anomalies rather than find them. This is not problematic, although increasing the ease of using those functions to find anomalies would make the prototype more user friendly by allowing several possible methods of using it. Though grouping log lines into message groups is expensive, it was beneficial enough for understanding the contents of the log files that it should be kept in future iterations of the prototype.

The irregularity of the log files from the Windows and Ubuntu machines lead to a large number of anomalies being found. Though all log lines described unusual events, their contents were most often of an informative nature, on which an alarm is not of interest to the user. The fact that days that were lacking log files lead to an absence in analysis and anomaly detection is a distinct flaw of the prototype. Future iterations of the prototype should assume that a missing log file can be interpreted as a completely empty log file and perform the normal procedure. This also counteracts the problem of frequent resetting of the analysis.

It is important to note that most errors were handled and classified accurately, as indicated by 11 out of 14 known errors being handled correctly. The vast majority of the erroneous handling consisted of false alarms. The prototype is accurate enough to be used as an indicator of possible issues, though it is not yet ready to be used as the sole way of monitoring a system.

A shift in the normality of a log file, as seen in Figures 6.14 and 6.15, is handled acceptably. It takes at least one week to adapt to spikes, due to performing comparisons based on weekday. An overall shift in the number of lines is adapted much faster, and though the exact speed depends on the size of the change, three days is usually enough. Though the skew of the data leads to faster adaptation, the observed visualization in a number of cases suggests that the adaptation should be even quicker. It is however important to keep in mind that too quick adaptation can lead to errors not being detected, as was seen in Section 6.5.

If there is no analyzed data to compare a day to, the prototype assumes that there is no history, and resets the analysis by treating the current day as the first. If a single day of data happens to have been lost, whilst still retaining the rest of the history, this would lead to suboptimal anomaly detection. As a countermeasure, it would be possible to look at older data and assume that the missing data was "normal". However, this could pose a range of problems, such as the possibility of normality changing during the missing day. Furthermore, if a day's worth of data disappears, it is very likely that it is a problem to which the user would like to be alerted.

One thing to note is the necessary period of data to perform analysis. If comparisons are to be done on a certain day, data from one week earlier must be available. If not, the analysis simply resets. Overall, it can also be said that the analysis is better after a period of time, as two weeks of data only provides two point of data to compare.

One issue with the anomaly detection is the danger of marking too many anomalies. Even though every anomaly marks a deviation from normality and should be investigated, too many anomalies may lead to users ignoring the anomalies. To avoid this saturation in alarms, different alert levels could be used to ensure that truly important anomalies are still seen by the user. This would however depend on being able to rank anomalies, something that is not currently possible.

The occurrences of out of memory exceptions severely limits the usability of the prototype. These occur during processing of the largest files (1-3 GB per day each), which would have benefited the most from being summarized and presented. The solution to this was to exclude steps that lead to this, for example grouping by message, while performing the rest of the analysis. Another solution to this issue would be to equip the machine running the prototype with more memory, or to limit the memory used in some way. This could be done by for example dividing the largest files into smaller chunks that are processed separately and then gradually combined to compile a summary, though that would lead to a largely increased processing time.

The most useful part of the analysis to spot anomalies was the graph over the total line counts. Not only did it show found anomalies in a clear manner, a lot of information about the expected state of the system could easily be conveyed. The message groups were the most useful tool to understand the anomalous contents of a certain log file. Note that due to assuming that statistical rarity is proportional to the likelihood of a fault, the prototype cannot be used to detect systematic errors.

## 7.5 Flexibility of data format

Due to the assumptions made, only line-based log files that are divided by day and start with a timestamp are guaranteed to be handled correctly. The tests on Windows 8.1 and Ubuntu 14.04 system logs showed that different formats could be handled well.

However, the format for setting the format is rather obtuse. In the settings file, the format must be input according to the conventions of C#'s DateTime conventions as listed in Microsoft (2015). For example, if the dates are of the format "Monday April 27 2015", the user must write "dddd MMMM d yyyy" in the settings file.

The current processing also assumes that the timestamp at the beginning of a line starts with the hour. This somewhat limits what timestamps can be used in the log files, especially concerning instances where the timestamp also includes the date. The timestamp settings are not very intuitive, as a basic knowledge of regular expressions is required.

## 7.6 The effect of anomalies on data

Once anomalies have been detected, it is not trivial to decide how to handle the data. As mentioned, abnormal data should not be used to make predictions (Hill and Minsker, 2010), and thus anomalies should not be included when creating the model of normality. However, this was found to be more complex than it seemed.

The first problem arises when an anomaly is found in aggregated data. Since the abnormal aggregated point cannot be excluded without causing an absence of data, the point must be normalized instead. This could be done by for example setting the aggregated value to an average, or to the closest acceptable value. Since the correct, "healthy" value is not known, any adjustments may lead to skewing the data in an incorrect way, which may be more damaging than helpful. The second issue is related to the presence of false positives. If for example changes in how logging is done triggers a false positive, adjusting the data will lead to future anomaly detection being affected, possibly triggering more false anomalies.

One approach to handling anomalous data when creating the model would be to let the user decide if every anomaly is a true anomaly, and then remove such data before normality is calculated. However, the user may not know if it is a false positive or not, and it places a large burden on monitoring the data on the user, something that contradicts the goals of the prototype.

Another way would be to create two historical files for that date: one that treats the anomaly as a false positive and adapts to the change, and one that treats it as an actual anomaly and instead uses some normalized data for that point. If the anomaly was to appear again on the next day, one could assume that it is not an anomaly but an intended change in the system. The file that adapted would then be kept as the correct historical file for the previous day, while the other file would be discarded. In other words, every file containing a possible anomaly would generate two paths on that day and pick which path to follow the next day. Generating two files per day would of course demand more space, but only until one of the files has been deleted.

The current version of the prototype does not exclude anomalies, due to the aggregated nature of the data. One resulting issue from this can be seen in Figure 6.9, where the correct Thursday and Friday spikes are marked as anomalous due to incorrect spikes occurring at the wrong hour the previous week. Future versions of the prototype should adopt a policy regarding how to normalize abnormal days, in order to not taint future anomaly detection.

## 7.7 Other issues

A point of interest found during the development is related to ethics. Log files contain sensitive information such as email addresses, home addresses and customer names, but rows containing such information is in the minority. Of course, all log rows may contain information that must be kept confidential due to business reasons, but this paragraph focuses on sensitive customer information specifically. Once the log files have been processed by the prototype, customer information can be very easily viewed. Though this prototype is intended for internal use, single customers may get a lot more exposure than they would when only mentioned in a rarely used log file. When the information is collected and then presented, it may become necessary to draw up guidelines for how the prototype should be used, and how data is shared. Increasing the exposure of sensitive information also increased the responsibility of handling it in an ethical manner.

Contrary to the modular flow envisioned in Figure 6.1, some tasks are more convenient to do during the “wrong” stage of processing. For example finding words from the blacklist, which is part of the anomaly detection, is best done during preprocessing when compiling a list of all occurring words. This is in general due to the fact that reading large files is expensive in terms of time. Keeping the prototype modular would have simplified future development, but doing so sometimes result in a loss in performance. As performance was valued higher than modularity, the slight mix in workflow was allowed.

## 7.8 Future work

From the previous discussion, possible points of improvements of the prototype will be summarized. In general, there is much functionality that can be added or improved during future iterations, and a few different suggestions will be listed here.

The processing time was good, and the amount of storage space used was acceptable for the prototype. Future iterations should above all focus on greater exactness in the anomaly detection and the view of normality as they were the weakest areas of the prototype. The total word and bigram counts, and the individual bigram counts could be used perhaps to calculate a score regarding the change in the file. Work should be done to reduce the number of false positives, and to further improve the grouping of messages.

Another priority should be to prevent the processing machine from running out of memory. This may include a new way of structuring the files, but it is clear that it should be possible to run all stages of processing on all available log files.

One area of improvement that would be important for the usability of the prototype is increased control to the user. Adjusting alert levels or dismissing found anomalies would also be useful when the user knows the domain well enough to be able to spot false alarms. To further enable the user to affect the alerts, a whitelist could be added. Words in the whitelist would always be interpreted as benign and never raise alerts. They would however count towards the total number of words, and thus contribute to an anomaly.

The existing blacklist could be improved by allowing the banning of entire phrases, not only words or bigrams. Being able to dynamically update the blacklist, or having a separate search function, would assist users in finding the cause of errors.

As a way to gain even more insight into what caused an anomaly, it would be useful to be able to directly link an anomaly to the log line where the error has occurred. However, a single log line rarely serves as a foundation for an alert. Most often, it is an aggregated value that is the cause. Even so, linking the prototype to the actual log files in some manner may improve the understanding of the errors.

Currently, all old log files are saved until they are manually removed. This was useful during development, but if the prototype is to actually save hard drive space, old log files should be deleted. This is today done by 3bits, but if the prototype is to completely manage log files, it must be able to perform this task. Future work should pinpoint exactly how long it is useful for old log files to be stored, and then remove them. It is also possible that the analysis should be removed after a sufficient period of time.

In terms of the visualization, a monthly and a yearly view over total counts of the data may be useful, depending on the duration of storage. A top-level indication of interesting files should be added, and the list of log files should be divided into sections based on originating folder. For the graph over total counts, a legend explaining the different elements for the graphs should be added. The view for message groups should be clarified by changing the heading to “Anomalous message groups” and labeling the hours, and other ways of visualizing the messages should be explored. Finally, tooltips could be added in a number of places to help clarify the interface.

# 8

# Conclusions

This thesis has resulted in a prototype for automated log analysis, targeted towards textual log files that are line-based. It was able to accurately detect 11 out of 14 known errors, and adapted well to a shift in normality. This shows that it is possible to perform automated log analysis without making assumptions about the data, or requiring human input.

From the discussion in Chapter 7, it is possible to draw the conclusion that the accuracy of the anomaly detection must be further increased, due to the risks associated with marking too many anomalies. Having fewer false alarms would increase the usability and trustworthiness of the prototype.

The prototype is better suited for systems that consistently produce log files, due to the analysis performing better when historical data is available. Log files that are consistent in what and when they output gain a more accurate anomaly detection, due to the inherent patterns in the files. Small log files do not benefit from being analyzed with respect to storage used, as the overhead for the analysis is larger than the files themselves.

Users found the prototype useful, and could describe scenarios where it could assist them in their work. The prototype could aid in making log handling more proactive, and reduce the time to locate errors. Except for minor adjustments to the interface, an overview of all log files at once was found to be needed, as well as a way to search for occurrences of specific words. Some elements in the visualization, such as bigrams, were not used, which indicates that the interface can undergo further polishing.

# Bibliography

Bolton, R. J. and Hand, D. J. (2001), 'Unsupervised Profiling Methods for Fraud Detection', *Proc. Credit Scoring and Credit Control VII* pp. 5–7.

Bostock, M. (2012), 'D3.js - data-driven documents', `http://d3js.org/`. Last accessed: February 26, 2015.

Chang, C. H. and Ding, Z. K. (2005), 'Categorical data visualization and clustering using subjective factors', *Data and Knowledge Engineering* **53**, 243–262.

Chen, M.-S., Han, J. and Yu, P. S. (1996), 'Data mining: an overview from a database perspective', *Knowledge and data Engineering, IEEE Transactions on* **8**(6), 866–883.

Chuvakin, A. (2013), *Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management*, Syngress, Waltham, Mass.

Esmaeili, M. and Almadan, A. (2011), 'Stream data mining and anomaly detection', *International Journal of Computer Applications* **34**(9), 38–41.

Gantz, J. and Reinsel, D. (2011), 'Extracting value from chaos', `http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf`. Last accessed: Febuary 23, 2015.

Ghoting, A., Parthasarathy, S. and Otey, M. E. (2008), 'Fast mining of distance-based outliers in high-dimensional datasets', *Data Mining and Knowledge Discovery* **16**, 349–364.

Google (2015), 'About intelligence events', `https://support.google.com/analytics/answer/1320491`. Last accessed: January 7, 2015.

Gunter, D., Tierney, B. L., Brown, A., Swany, M., Bresnahan, J. and Schopf, J. M. (2007), Log summarization and anomaly detection for troubleshooting distributed systems, *in* 'Grid Computing, 2007 8th IEEE/ACM International Conference on', IEEE, pp. 226–234.

Hay, A., Cid, D. and Bray, R. (2008), *OSSEC host-based intrusion detection guide*, Syngress Pub, Burlington, Mass.

Hill, D. J. and Minsker, B. S. (2010), 'Anomaly detection in streaming environmental sensor data: A data-driven modeling approach', *Environmental Modelling and Software* **25**(9), 1014–1022.

Hodge, V. J. and Austin, J. (2004), 'A Survey of Outlier Detection Methodologies', *Artificial Intelligence Review* **22**(1969), 85–126.

Jackson, P. and Moulinier, I. (2007), *Natural Language Processing for Online Applications: Text retrieval, extraction and categorization*, John Benjamins Publishing, Philadelphia.

Jacobs, A. (2009), 'The pathologies of big data', *Communications of the ACM* **52**(8), 36–44.

Jayathilake, D. (2012), 'Towards structured log analysis', *JCSSE 2012 - 9th International Joint Conference on Computer Science and Software Engineering* pp. 259–264.

Jones, K. (2013), 'Fuzzystring', `https://fuzzystring.codeplex.com/`. Last accessed: April 23, 2015.

Knox, E. M. and Ng, R. T. (1998), 'Algorithms for Mining Datasets Outliers in Large Datasets', *24th International Conference on Very Large Data Bases* pp. 392–403.

Lee, W., Stolfo, S. J. and Mok, K. U. I. W. (2001), 'Adaptive Intrusion Detection: A Data Mining Approach', pp. 533–567.

Logentries (2015), 'Logentries', `https://logentries.com/`. Last accessed: January 7, 2015.

Microsoft (2015), 'Custom date and time format strings', `https://msdn.microsoft.com/en-us/library/8kb3ddd4%28v=vs.110%29.aspx/`. Last accessed: May 7, 2015.

Sample, C. and Schaffer, K. (2013), 'An Overview of Anomaly Detection', *IT Professional* **15**(1), 8–11.

Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D. and Tufano, P. (2012), 'Analytics: The real-world use of big data', *IBM Global Business Services Saïd Business School at the University of Oxford* pp. 1–20.

Splunk Inc. (2015), 'Splunk', `https://splunk.com/`. Last accessed: April 24, 2015.

Srivastava, N. and Srivastava, J. (2010), 'A hybrid-logic approach towards fault detection in complex cyber-physical systems', *Annual Conference of the Prognostic and Health Management Society* pp. 1–11.

The Stanford Natural Language Processing Group (2015), ‘Stanford tokenizer’, `http://nlp.stanford.edu/software/tokenizer.shtml/`. Last accessed: May 12, 2015.

Veasey, T. J. and Dodson, S. J. (2014), ‘Anomaly Detection in Application Performance Monitoring Data’, *International Journal of Machine Learning and Computing* **4**(2), 120–126.

Vesanto, J. (1999), ‘SOM-based data visualization methods’, *Intelligent Data Analysis* **3**, 111–126.

Ware, C. (2013), *Information visualization: perception for design*, Elsevier, Waltham, Mass.

Yamanishi, K. and Maruyama, Y. (2005), ‘Dynamic syslog mining for network failure monitoring’, *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD ’05* p. 499.