UTRECHT UNIVERSITY

MASTER THESIS

Anomaly Detection in Application Log Data

Author:
Patrick KOSTJENS

First supervisor:
Dr. A.J. FEELDERS

Second supervisor:
Prof. Dr. A.P.J.M. SIEBES

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computing Science

in the

Algorithmic Data Analysis Group
Department of Information and Computing Sciences

August 2016

ICA-3733327

UTRECHT UNIVERSITY

Abstract

Faculty of Science
Department of Information and Computing Sciences

Master of Science in Computing Science

Anomaly Detection in Application Log Data

by Patrick KOSTJENS

Many applications within the Flexyz network generate a lot of log data. This data used to be difficult to reach and search. It was therefore not used unless a problem was reported by a user. One goal of this project was to make this data available in a single location so that it becomes easy to search and visualize it. Additionally, automatic analysis can be performed on the log data so that problems can be detected before users notice them. This analysis is the core of this project and is the topic of a case study in the domain of application log data analysis.

We compare four algorithms that take different approaches to this problem. We perform experiments with both artificial and real world data. It turns out that the relatively simple KNN algorithm gives the best performance, although it still produces a lot of false positives. However, there are several ways to improve these results in future research.

Keywords: Anomaly detection, data streams, log analysis

Acknowledgements

First and foremost, I would like to express my gratitude to Dr. Ad Feelders for all his feedback, advice and reviews throughout the process. Furthermore, I would like to thank Prof. Dr. Arno Siebes for being the second supervisor of this thesis.

I would also like to thank everybody at Flexyz and in particular Iwan Faber and José Hilgenkamp for making my internship there possible and their help with finding a topic for my research. I also would like to especially thank everybody that helped me gather all the data that was needed for this project.

Finally, I would also like to thank my family, friends and girlfriend for their mental support throughout this project.

Contents

List of Figures

List of Tables

List of Abbreviations

CBLOF	**C**luster-**B**ased **L**ocal **O**utlier **F**actors
ELKI	**E**nvironment for deve**L**oping **K**DD-applications supported by **I**ndex-structures
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
KNN	**K**-**N**earest **N**eighbours
LOF	**L**ocal **O**utlier **F**actors
LRD	**L**ocal **R**eachability **D**istance
MAP	**M**aximum **A** **P**osteriori
MCMC	**M**arkov **C**hain **M**onte **C**arlo
MMPP	**M**arkov **M**odulated **P**oisson **P**rocess
TN	**T**rue **N**egative
TP	**T**rue **P**ositive

Chapter 1

Introduction

Modern applications and servers can produce big amounts of log data. As the number of servers and applications grows, the amount of log data that is produced quickly becomes too big to analyze manually. Additionally, most of the records in this data are not interesting, because they are part of normal operation. However, the records that are interesting are not easily found because there are relatively few of them. We still do want to find these records though since they can indicate, for example, problems with an application or server.

1.1 Motivation

This thesis research is executed as an internship at Flexyz. Every thesis has a motivation from a scientific perspective. However, due to the fact that this thesis research is done in the form of an internship, there is also a direct motivation from a business perspective. We will start by looking at that business perspective, followed by the scientific motivation.

1.1.1 Business motivation

There are a number of reasons why we might consider a log record or a group of log records to be interesting:

- A message is new and we never saw it before.
- A message appears more often than usual.
- A message appears less often than usual.
- A message stops appearing.

Messages matching any of the above descriptions can be considered to be anomalies and are therefore potentially interesting. Because there are a lot of messages in the logs and we only want to know about the interesting ones, automatic analysis of those messages would greatly reduce the amount of manual work needed. An additional benefit of automatic analysis is that it can be pro-active. This means that an automatic analysis tool can constantly monitor the logs and report interesting messages as soon as they occur. This has a big advantage, namely that problems can be detected much faster. Without such a tool, the logs would only be searched when there is a known problem about which a user wants to find more information. This also means that problems would be detected by the user of an application or server, while in the case of automatic analysis of the logs, problems can be detected and possibly solved before a user suffers from them.

Data mining techniques may provide a solution that can be used by such a tool. Those techniques need to be unsupervised, because it is infeasible to create a labeled set of log records.

1.1.2 Scientific motivation

Aside from the business perspective, there also is a scientific motivation. There are some fields within anomaly detection that have received some special attention, like intrusion detection [19] and fraud detection [8]. A domain that has received less attention is that of log data analysis. Algorithms that are focussed on a different domain are not necessarily useless for our domain, but they may need some adjustments. The different domains can use some of the same general algorithms, but the priorities may be different. For example, in intrusion detection, it is required to immediately block intrusive requests, and therefore the algorithm should be able to determine whether something is an intrusion or not very quickly. In fraud detection however, it may not be a problem if this takes some more time, but it may be more problematic if a case of fraud is missed.

In all domains a similar consideration has to be made. Since anomaly detection algorithms do not have perfect accuracy, generally a balance between false positives and false negatives has to be found. If the algorithm is more strict, more false positives (messages marked as anomalies while they are not) may occur, while a more loose algorithm might have more false negatives (undetected anomalies). When determining how to balance these two errors for a specific domain, the severity of both a false positive and a false negative has to be taken into account. In the case of log data analysis, a false positive will generally result in unnecessarily spent man hours. A false negative may result in a problem with an application or server that is detected later on by a user, which can cause dissatisfaction or possibly data loss. Within this domain it may even depend on the application which is worse.

Another way the human workload can be reduced is by showing enough context when an anomaly is detected. If we can, for example, show that a message was classified as an anomaly because it started occurring more or less often, this can help when investigating the problem. An additional improvement would be to group messages that jointly showed the same deviation from the usual pattern into a single report so that there is more detail available about the problem. This can prevent or reduce the need to research whether multiple reported anomalies are related and are indicators for the same problem. Some of the points mentioned above are not only relevant from the scientific perspective, but also from the business perspective.

1.2 Problem statement

We already looked at a brief definition of the problem in the previous section. In this section, we will discuss the definition of the problem and some properties of the problem in more detail.

In the motivation, we already mentioned some reasons for an anomaly to be identified as such. Two of these reasons were already pretty clear. A new message we did not see before or a message that stops appearing is relatively easy to detect. However, determining when to report a message that appears more or less often than usual is a bit more difficult. Since some variation in the number of log records being written is expected for our application domain, we do not want to report an anomaly if there is one extra message while there usually are a million messages in a certain time interval. Although the threshold for this kind of anomaly cannot be an absolute number or percentage that is chosen beforehand, we can make this a bit more concrete.

If we look at previous log records, we can determine which log records are a different report of the same messages. Over these messages, we can, for example, calculate the average over a certain time window. Naturally, some types of log records may show more variation than others. Therefore, a fixed percentage of allowed deviation from the average will not work properly. What we can do, however, is calculate the mean and standard deviation over a certain period of time. We can then determine the bounds on the variation by taking the range from, for example, the mean plus or minus twice the standard deviation. If the number of recorded log records then falls outside this bound, we report an anomaly. The exact bounds may still require tuning, which will be done during the experiments.

It may seem like this definition of an anomaly is almost enough to solve the problem with a relatively simple algorithm. However, the amount of log records we expect during the day, for example, may be different from the amount of log records during the night. Since a mean and standard deviation cannot handle these kinds of patterns, we need more advanced algorithms that find these patterns so that we know to what part of the data new observations should be compared. If we know what parts of the data we expect the new data to match, we can use the idea described above. Note that some or most of the algorithms we are going to discuss may have their own way to set a boundary between anomalies and expected data. Such algorithms will not necessarily use this idea, but something similar to prevent setting an absolute boundary.

1.3 Challenges

Some challenges have already come up in the previous sections. An extra challenge that may be of scientific interest is that it is likely that there will be gaps in the data. If the server that gathers all the logs is temporarily unavailable or there is a network disruption, the log messages might get lost for a certain amount of time for one or more servers. More reasons that cause loss of log messages exist so gaps in the data are a near certainty. The difference between these kind of gaps and complete downtime of a certain server may be hard to detect. However, for the intended purposes of the application, we want the tool to report that there is a problem when there is a gap in the logs. Additionally, we do not want the classification of future log messages to change significantly because of such gaps, or any other anomalies for that matter.

Another challenge, that has briefly been discussed above as well is the fact that the amount of log data that has to be processed can be enormous. This brings us challenges on multiple fronts. First of all, we have to be able to perform the anomaly detection on the data fast enough, otherwise we will not be able to report anomalies in real-time. A second problem is that we can only keep a limited history of the log data before running out of disk space. This means that we need algorithms that can work with some aggregated form of the data if we want to be able to use more than a couple of days of history as we will see in Chapter 2. The last challenge related to the scale of the data is the fact that we cannot load (nearly) all the data in memory for processing. This once more means that the algorithms either need to be able to work with an aggregated form of the data or that they need to be rather memory efficient as well aside from the required running time efficiency.

Depending on the knowledge of the data our algorithms have, another possible challenge might be caused by the fact that not all log data is accurate. As we will see later, the data, for example, includes a field with the log level for a log record. This indicates the severity of the message being logged. However, these severities are not always accurate. This problem has been confirmed by a domain expert, so any algorithm that attaches meaning to this information might be influenced by the inaccurate data.

1.4 Research questions

The motivations, problem and challenges discussed in the previous sections lead us to the following research questions:

(1) *Can on-line anomaly detection be performed on log data while limiting the number of false positives?*

(2) *Can the anomalies be placed in context so that we can clearly state why they are classified as anomalies?*

(3) *Can the anomaly detection model handle changes over time without future predictions being influenced by incidental anomalies?*

The first research question can be answered by conducting experiments with the different algorithms that will be discussed. The second question may depend on the algorithm, and may also require some new method to be developed to identify messages that belong together. Therefore, it will require some extra research and experimentation aside from the implementation of the algorithms that are going to be discussed.

The third question is quite difficult to answer. Although one might expect that this can be answered by creating some artificial datasets that test this behaviour and looking at the results the different algorithms give for these datasets, this would probably not give very representative results. Since we do not know what kind of changes in the application behaviour are present in the real world data, it is not possible to create artificial datasets that reliably model this behaviour. Aside from the way to find the answer, we may be able to achieve the desired behaviour by removing anomalies from the dataset after they have been reported. However, some algorithms may already be showing the desired behaviour and do therefore not require modification. One danger of removing the anomalies from the dataset after detection is the fact that changes over time may not be found, therefore it may be necessary to store these anomalies elsewhere so that they can be incorporated into the dataset again when they seem to be forming a new or changed pattern. This does make the changes more difficult though and still only prevents future predictions from being influenced.

1.5 Outline

We will start the rest of the thesis by looking at some background information. This, among other things, includes the way we gather log data, what log data we gather, and an overview of related work. Next, we discuss the algorithms we use in the experiments and the tools we use to implement these algorithms. After that we take a look at the experiments that were performed and we will evaluate the results. Finally, we briefly summarize our findings in the conclusion.

Chapter 2

Background

In this chapter, we will be discussing some background information regarding this case study. This information will give some more insight into the problem domain. We will do this by discussing what log data is available and how we gather and process it. Furthermore, we will look at related work that might be relevant to this research.

2.1 Log data

The background information regarding the log data will be split into two parts. First, we will discuss the data that is available for our research. After that, we will take a look at how we acquired that data.

2.1.1 Available data

For every log record, we have a lot of data available. We may not use all of the data in the log process and will only discuss the relevant properties here. The most important field for the user that handles the reports is the *message* field. This contains the actual message that was logged and therefore can give a lot of information about the possible problem. However, this field is not really used in our analysis.

As we already discussed briefly, our analysis may depend on finding all the log records that log the same message at a different time or, for example, for a different user. Although the actual message may seem like a logical way to group log records, we have data that can make this far easier. Note that some information about the user may be present in the message. Therefore, the exact text of the message may be different, while we still want to group these kinds of log records together. A rather reliable way to group log records, is to group them based on the location in the code that logged the message. We have information like the name of the class, the file, and the line number where the log record was generated. By combining this information with the information about the application and the host it runs on, we can group all the log records that belong together per instance of an application. Note that no two applications with the same name run on the same server, therefore the combination of host name and application name uniquely identifies an application.

Although the content of the message in a log record is not required for analysis, we of course do keep track of it because it needs to be reported when reporting an anomaly. Aside from the fields we just discussed, some additional data is available that may help in the analysis. We also have log levels (i.e. debug, warning, error, etc.) and timestamps for every log record.

The log levels may be used to customize the thresholds for anomalies. For example, a certain deviation from the norm for errors may need to be reported sooner than would be required for warnings. However, this can later be added as a useful feature by the company, but it is not a part of our research. Timestamps are, of course, required to know when the messages were logged and are therefore essential for the analysis.

2.1.2 Data gathering

We split the explanation of the data gathering process into two parts. We start by discussing the general infrastructure used for the entire process. After that, we will explain the preprocessing of the data in more detail.

Infrastructure

At the start of the project, all the log files were simply stored on the disks of the servers they were generated on. However, this made it difficult to access and analyze the logs. Therefore, we needed to create a solution that gathers all these logs and stores them in a central location, which allows more easy access to the logs. The log messages should be transferred from the applications and servers to the central location with all the logs automatically. During normal operations, it takes at most a couple of seconds for a log record to get from the application that generates it to the Elasticsearch cluster, so real-time analysis using the central Elasticsearch database is possible.

This process has been devised and implemented during the first phase. This setup started gathering logs from multiple applications within the Flexyz network at the beginning of April. Therefore, plenty of log data from different production servers is available for our experiments. The process uses the ELK[1]-stack. The ELK-stack is a combination of Elasticsearch, Logstash, and Kibana. It is commonly used to make log data, but also other unstructured data, easily accessible and should not be confused with the unrelated ELKI framework we will be discussing later.

A visualization of the used infrastructure can be found in figure 2.1. For the servers that are newly created, some rough specifications are outlined in the boxes representing those servers. Note that these specifications can change when required. Also note that the additional box behind the Elasticsearch server means that another server with the same tasks and specifications will be used. Those servers will serve as an Elasticsearch cluster. The dotted lines represent requests by users and are not part of the analysis process. The other lines represent connections that are part of the analysis process.

The process starts with log messages that are generated by applications on the servers in the bottom of the image. Log messages can also be generated by the servers themselves. In both cases, the systems logging utility, called rsyslog, will be used to send all the logs from the server (and application) to the proxy in the middle. Note that other servers and applications that cannot use rsyslog can also be added later on, but rsyslog was chosen as the initial tool because it is already present at the majority of the servers.

[1]See: https://www.elastic.co/webinars/introduction-elk-stack

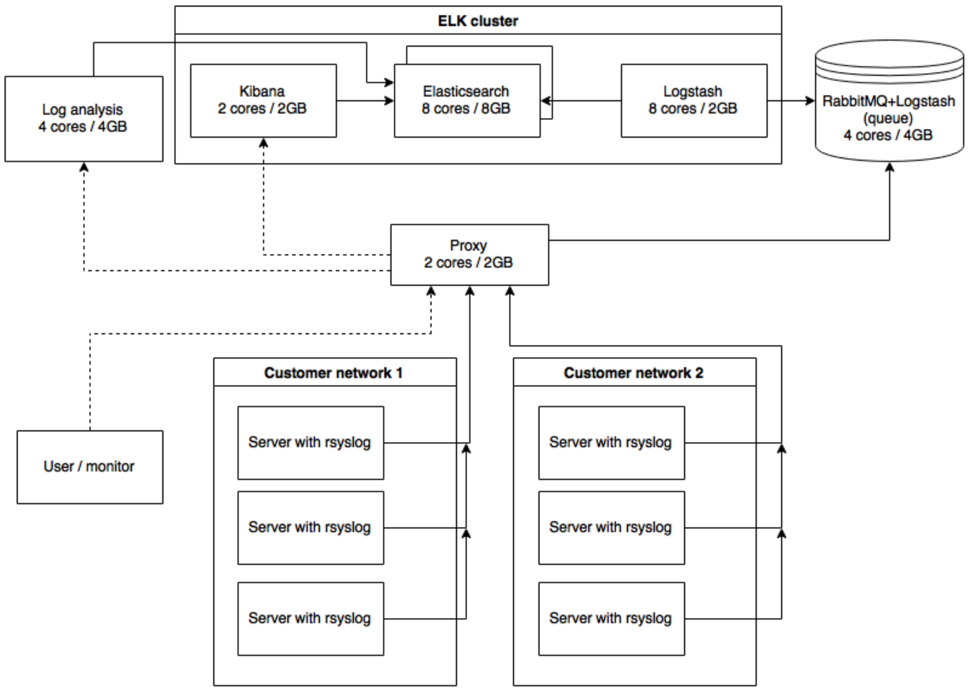

FIGURE 2.1: An overview of the infrastructure used for gathering and analyzing logs.

Of course far more servers and customer networks can exist than depicted in the figure.

The proxy forwards all the logs that it receives to the first Logstash server (on the right). This Logstash server also contains a RabbitMQ instance. Logstash will store all incoming messages in this queue. The second Logstash server will then read the messages from the queue and extract all the required information from them. It then stores the messages in Elasticsearch. The RabbitMQ queue is added to prevent loss of log messages in case the Logstash server occasionally cannot process the messages fast enough.

When the log messages are persisted in Elasticsearch, several applications can read the messages that are stored there. One of those applications is Kibana. This tool can create visualizations and summaries of the data in Elasticsearch. It is not essential to our research, but it can help exploring the data that is available. It is also used by other people within the company to do some ad hoc analysis of the data in Elasticsearch.

The server that will run the tool based on the research in this thesis project, is the log analysis server. This server also reads its data directly from Elasticsearch and stores it in an aggregated form. It will then analyze the messages and serve the anomalies that are found using a web interface. This web interface can be reached by users through the same proxy that is used for incoming messages. The same idea regarding the proxy holds for Kibana as well.

Preprocessing

Some preprocessing is needed to get the log data from the applications that generate the data to a database in which we can easily search it. This process is related to the infrastructure we just discussed, because the different servers in that infrastructure perform different operations on that data. First, we will discuss the process for the log data that is generated by the applications. After that we will look at the process for log data generated by the servers. The difference between the two is not very big, but nevertheless important.

The majority of the applications at Flexyz is written in Java. For these applications, we add a library that sends the log data that is generated to rsyslog in a JSON format. Note that rsyslog is the software used for logging and, in our case, transferring the data. This data includes quite a bit of information about the message being logged. The following fields are recorded:

- *message*: The text of the record that is being logged.
- *line_number*: The number of the line in the Java code that generated the message.
- *class*: The Java class in which the message was generated.
- *thread_name*: The name of the thread that logged the message.
- *method*: The name of the Java method that logged the message.
- *level*: The log level of the message (i.e. debug, error, warning, etc.).
- *logger_name*: The name of the Java logger that logged the message.
- *file*: The name of the Java file that logged the message.
- *exception*: Only logged in case an exception occurred.
  - *exception_class*: The Java class name of the exception that was thrown.
  - *exception_message*: The message of the exception that was thrown.
  - *stacktrace*: The call hierarchy to the point in code where the exception was thrown.
- *timestamp*: The date and time at which the message was generated.

Next, rsyslog creates a new JSON object for every record that is received from the application. In this object, the originally received object is added, as well as the name of the host the application is running on. The name of the application that generated the log message is also added to this object. This object is then sent to the proxy server, which in turn forwards it to the first Logstash instance.

This Logstash instance then puts the object in the RabbitMQ queue as discussed in the previous part about infrastructure. The second Logstash instance, which reads the data from the queue, then continues the processing. It first decodes all the JSON. Next, it parses the timestamp of the message into a format that can be used more efficiently for storage and querying. It also flattens the object so that all the information from the Java application (including the exception if it is present) is available at the same level in the object as the host and application name. Finally, it makes all the text in the *level* field lowercase, so that it becomes uniform across all applications. The fully processed log record is then sent to the Elasticsearch cluster, so that it can be queried easily by the analysis tool and other applications.

The process for other applications, including system processes, is not much different. However, those log records contain less information. Although these records do have a *message*, *level*, and *timestamp*, they have none of the other fields we listed above. As for the messages from the Java application, a host and application name is added to these records as well. The rest of the process is the same, except for the fact that the timestamp generated by the other applications is different and therefore will be parsed in a different way.

As we mentioned in Section 2.1.1, the data will be grouped using the *host*, *application*, *class*, and *line_number*. The aggregations we create are time based. For example, to compress the data, we can record the counts of each group of messages per 5 minutes. This approach allows us to store the data for a longer time. Note that the specific aggregations we need may depend on the algorithm. However, this general aggregation will be used so that we can hopefully prevent having to completely throw away data because we are running out of storage. If other aggregations cannot be deduced from this aggregation, they will simply have to be made over a shorter time period.

2.2 Related work

Anomaly detection is a field within data mining that has already developed techniques that are applicable to the sort of problem we are considering here. The field is known under a few different names, including outlier detection, novelty detection, noise detection, deviation detection, outlier mining, and exception mining [14]. However, not all the techniques for anomaly detection are applicable due to the specific properties of our domain as we discussed before. We already mentioned that some specific applications of anomaly detection (e.g. fraud detection, network security) have received extra attention [8, 19]. Although the techniques presented there may be usable in our domain as well, we will use general anomaly detection techniques as the starting point.

Some very useful surveys have already been published that compare many different algorithms [6, 7, 11, 14, 18, 19]. These surveys present a good overview of the different types of algorithms that are available and are useful when deciding what algorithms to use. However, they do not actually implement the algorithms and compare their performance.

The surveys generally group the algorithms by their approach. These approaches include clustering-based, nearest neighbour-based, information theoretic, classification-based, and temporal approaches. We will discuss most of these approaches in the next chapter, but not all of them can be used in our domain. Since our domain requires an unsupervised method, we cannot use information theoretic and classification-based methods as those methods require labeled training data. The other approaches can be used and we use the information in these surveys as a starting point for our research.

Aside from different approaches, another important topic that is discussed in several papers is the definition of anomalies and what types of anomalies exist [6, 7]. Chandola et al. distinguish three types of anomalies, namely point anomalies, contextual anomalies and collective anomalies. Point anomalies are anomalies where a single record is considered to be an anomaly in a global context. A contextual anomaly is the same, except that the point is only considered to be an anomaly in a smaller, local, context while it is not considered to be anomalous in a global context. Collective anomalies are caused by the presence or absence of groups of records that usually occur together. Throughout this thesis, we will be looking at point and contextual anomalies.

Chapter 3

Algorithms

To be able to answer the research questions posed in Chapter 1, we will implement several algorithms in an attempt to solve the problems these questions pose. The algorithms we implement, and the way we implement them will be discussed in this chapter. Before discussing the algorithms themselves, we will have a look at the ELKI framework [23]. This framework can help make the implementation of some of the algorithms easier and also make it easier to compare the results of different algorithms. After discussing ELKI, we will go over the different algorithms we are going to compare. We discuss the details of how the algorithms work, as well as any relevant implementation details.

The algorithms we discuss can be subdivided into different categories in several ways. First of all, we can group them based on their problem approach. We discuss algorithms with clustering-based approaches as well as distance-based, nearest neighbour-based, and evolving prediction approaches. Another way to subdivide the algorithms is to split them into a group that uses the temporal aspect of our data and a more general group that does not. In this chapter, we first group the algorithms based on this temporal aspect. Within the two groups, we subdivide the algorithms into groups based on the approach to the problem that is taken.

3.1 ELKI

ELKI is an open source project that attempts to make the implementation and comparison of many algorithms in data mining easier [23]. Although it contains algorithms from several areas within data mining, it focuses on unsupervised methods in outlier detection and cluster analysis. The algorithms for outlier detection are, of course, the most relevant for this research.

The use of this framework can help us in several ways. First of all, a lot of algorithms have already been implemented, which means we do not need to implement them all ourselves. This saves time that can be spent on other parts of the research, like comparing more algorithms, and can prevent mistakes in our implementations that could affect performance. Additionally, the performance comparison between different algorithms is more fair since the algorithms share a lot of code and have similar overhead. This results in a far more comparable performance than would be the case with implementations from different people. A third advantage is the fact that the results should be easier to reproduce since the implementation of the algorithms is known and freely available.

ELKI has been designed as a modular framework. This means that components can easily be replaced. As a result, we can change a lot about an algorithm without changing the implementation of that algorithm. With ELKI it is, for example, easy to switch the distance function of an algorithm thereby making the algorithm more flexible and possibly usable with more datasets. Another example of this advantage can be found in algorithms that work using some clustered version of a dataset. With ELKI, such algorithms can easily be combined with many different clustering algorithms which can give better results.

ELKI also has a disadvantage. It is built so that you give a dataset as the input and, in our case, get anomaly scores and some other information as the output. However, some of the algorithms we will be seeing in the next sections do not fit this model. Streaming-based algorithms in particular will not be able to fit in the ELKI model. The general idea behind these algorithms, as we will see later, is to keep a model that changes over time. This model is updated using new data as it arrives and is used to predict for those new records whether they are outliers or not. Although such an algorithm could be implemented in ELKI, it would not make much sense since ELKI gets a static dataset as its input and forgets everything once it has computed the results. Implementing such a streaming algorithm would therefore mean that all the data would have to be read again when new data is added. This would defeat the purpose of the algorithm and we will therefore not implement such algorithms in ELKI. To determine the performance of an algorithm in our experiments, the streaming part is not required. So, although it may be faster when actually using the algorithms to use some streaming algorithm, it is not necessary for our experiments as long as the results are the same.

3.2 Non-temporal approach

We will start by discussing the algorithms that take a general approach. These algorithms do not attach any special meaning to the temporal aspect of the data and treat it like any other feature.

3.2.1 Nearest neighbour-based algorithms

The common idea behind nearest neighbour-based algorithms for anomaly detection is to compute an anomaly score using some relation (for example, the distance) to a number of nearest neighbours. We will start the discussion with the KNN outlier detection algorithm by Ramaswamy et al. [21]. After that we will look at a LOF algorithm by Breunig et al. [4, 5].

K-nearest neighbours

In 1998, Knorr and Ng proposed an anomaly detection algorithm that used a simple and intuitive definition [16]. Their definition considers every point that has no more than k points within a distance d to be an anomaly. In their definition, k and d are user defined parameters. The paper by Knorr and Ng was the basis for the first K-nearest neighbour (KNN) anomaly detection algorithm by Ramaswamy et al. in 2000 [21].

The changes to the original idea that were made by Ramaswamy et al. were made to solve a few important shortcomings. The disadvantages Ramaswamy et al. identified were the following:

1. The user has to specify a distance, d, which can be difficult to determine.
2. There is no ranking among the anomalies.
3. The algorithm is too inefficient to scale to higher numbers of dimensions.

Especially the first two points can be a problem in our application domain. This is the reason we will experiment with the algorithm by Ramaswamy et al. rather than the algorithm by Knorr and Ng.

Before looking at the solution Ramaswamy et al. have for the problems mentioned above, we will first discuss the algorithm they came up with. They built their algorithm based on the following definition of an anomaly: "*Given a k and n, a point p is an outlier if no more than $n-1$ other points in the dataset have a higher value for D^k than p*". In this definition, D^k is the distance to the k-th nearest neighbour of a point. This distance, D^k, is used as the anomaly score for this algorithm.

The definition that is given requires two parameters for the algorithm, namely, k and n. k indicates the number of neighbours that is taken into account, while n specifies the number of anomalies that has to be found. However, setting the expected number of anomalies beforehand may not be desirable. We will therefore work with a ranking of all the points based on their anomaly scores. Based on the relative anomaly scores that are found, we can then determine which of the points are outliers and which are not. We can, for example, separate the anomalies from the normal data by putting a boundary between the two at twice the mean anomaly score. This is just an example, and the experiments will determine what kind of threshold works well.

There is a trivial way to convert the above definition into an algorithm. The simple algorithm Ramaswamy et al. propose is a block nested-loop algorithm which has a running time that is quadratic in the size of the input. Alternatively, they also proposed two more efficient algorithms to find anomaly scores matching the above description. One uses an R*-tree to perform KNN queries, while the other partitions the data and prunes partitions that cannot contain anomalies thereby reducing the number of points to check. A disadvantage of this second algorithm is that it does not produce a complete ranking and therefore the idea we discussed above cannot work. Since the version of the algorithm that uses an R*-tree is already implemented in ELKI, we will use that for our experiments and discussion.

The pseudocode for the indexed algorithm as it is implemented in ELKI can be found in listing 1. This is not exactly the same code as Ramaswamy et al. proposed. An important difference is the fact that we produce a complete list of all points with their anomaly scores instead of returning only the anomalies (i.e. the top n). The outliers are than selected using a different threshold than a predetermined constant. Another important difference is that the implementation within ELKI can easily switch between different indexes, including R-Trees, M-Trees, k-d-trees, and also linear scans. When using the linear scan, the algorithm is roughly equal to the simple quadratic implementation we mentioned before.

Algorithm 1: Pseudocode for indexed KNN algorithm [21].

```
Input  : Positive integer $k$, dataset $D$
Output: Anomaly scores for all points in $D$
1 $index$ ← BuildRStarTree ($D$)
2 $results$ ← []
3 for Point $p \in D$ do
4     $distance$ ← FindKthNearestNeighbourDistance ($k$, $p$, $index$)
5     Add $distance$ for $p$ to $results$
6 end
7 $results$ ← SortDescending ($results$)
8 return $results$
```

Ramaswamy et al. solved the first problem by setting a fixed number of anomalies that have to be found instead of setting a fixed distance. The algorithm we give above, which is also the way it is implemented in ELKI, also removes the need to specify this parameter. The first problem is thereby solved completely, although some threshold above which points are marked as anomalies still has to be chosen. However, this is the case for all the anomaly detection algorithms. The second problem is also solved by both the original proposal and the pseudocode we gave. Since the distance to the k^{th} neighbour is used as the anomaly score, a ranking is automatically possible. Whether the third problem is solved, depends on the index structure that is used. However, in the case of both the proposed R*-tree and the simple quadratic approach, the algorithm scales well with the number of dimensions.

Since the publication of this algorithm, many improvements have been proposed. Among others Angiulli and Fassetti proposed a similar algorithm that works with streaming data [1]. Their algorithm should be able to give a performance improvement over the algorithm discussed above when working with streaming data, since it prevents analyzing the entire dataset again when new data arrives. Another possible improvement that also works with streaming data was proposed by Yang et al. [28]. Aside from these two, many more options have been proposed [2, 12, 24].

As discussed above, the indexed and the simple version of the algorithm by Ramaswamy et al. have already been implemented in ELKI. We will therefore use this publicly available version of the algorithm to make the experiments easier and more fair. Some of the alternative algorithms we just mentioned have also been implemented in ELKI. Therefore we can easily switch to an algorithm that improves some possible problems of the original algorithm.

Local outlier factors

Based on their own paper from 1999, Breunig et al. proposed the Local Outlier Factor (LOF) algorithm in 2000 [4, 5]. While previous nearest neighbour-based algorithms had problems with varying densities within the dataset, the LOF algorithm was built with the idea to solve this problem in mind.

The general idea behind the LOF algorithm is to compute the degree to which a point is an anomaly with respect to some local region around the point, while previous algorithms generally computed this with respect to the entire dataset. This improves the ability of the algorithm to recognize anomalies in datasets with clusters of varying densities. The example Breunig et al. give, can be found in figure 3.1. In this example, most algorithms mark o_1 as an anomaly, but, without looking at the local context, o_2 is not marked as an anomaly because it is as close to C_2 as all the points in C_1 are to each other. In contrast, LOF is able to capture both o_1 and o_2 as anomalies, without marking the points in C_1 as anomalies.

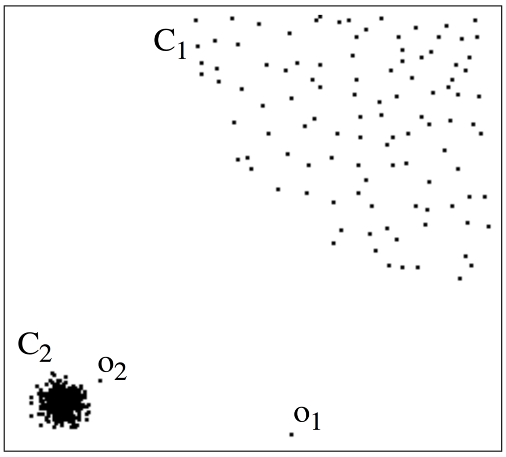

FIGURE 3.1: LOF use case by Breunig et al. [4].

The pseudocode for the LOF algorithm can be found in listing 2.

As can be seen in the pseudocode of the algorithm, the algorithm mainly consists of two steps. First, the Local Reachability Distances (LRDs) are computed. The LRD is a measure that indicates the closeness of a point to its nearest neighbours. In the second step, the LOF scores are computed. This score indicates the LRD of a point, compared to the LRDs of its neighbours.

With the KNN algorithm, we already saw that many different modifications for a single algorithm can be proposed. The same is the case for the LOF algorithm by Breunig et al. [4, 5]. Some authors modified the algorithm to make it more usable with streaming data [20]. Others have tried to improve the algorithm to make it usable in more or different scenarios [17, 26, 29]. Although many versions of this algorithm exist, we start the experiments with this original version. If a certain problem with the algorithm occurs (like performance problem, or a specific failure in detecting anomalies), we can always try to solve it by going over the possible improvements to determine if one of those may solve the problem. The original LOF algorithm by Breunig et al., as well as many of the proposed improvements have already been implemented in ELKI. This makes it easy for us to experiment with the algorithm, and if necessary switch to another version of the algorithm without much effort.

3.2.2 Clustering-based algorithms

The general idea behind clustering-based algorithms is that normal data instances are part of a cluster, while anomalies are relatively far away from

Algorithm 2: Pseudocode for LOF algorithm [4].

```
Input  : Positive integer $k$, dataset $D$
Output: Anomaly scores for all points in $D$
1  $LRDs \leftarrow []$
2  for Point $p \in D$ do
3      $sum \leftarrow 0$
4      $neighbours \leftarrow$ FindKNearestNeighbours ($k$, $p$)
5      for Neighbour $n \in D$ do
6          $kthDistance \leftarrow$ FindKthNearestNeighbourDistance ($k$, $n$)
7          $sum \leftarrow sum +$ max (dist ($n$, $p$), $kthDistance$)
8      end
9      if $sum > 0$ then $score \leftarrow |neighbours| / sum$
10     else $score \leftarrow \infty$
11     Add $score$ for $p$ to $LRDs$
12 end
13
14 $LOFs \leftarrow []$
15 for Point $p \in D$ do
16     $LRDp \leftarrow LRDs[p]$
17     $sum \leftarrow 0$
18     $neighbours \leftarrow$ FindKNearestNeighbours ($k$, $p$)
19     for Neighbour $n \in neighbours$ do
20         $sum \leftarrow sum + LRDs[n]$
21     end
22     $score \leftarrow sum/(LRDp \cdot |neighbours|)$
23     Add $score$ for $p$ to $LOFs$
24 end
25 return $LOFs$
```

those clusters. Some clustering-based techniques also take cluster density into account to determine more accurately whether a point is part of a cluster. We are going to discuss the cluster-based local outlier factors algorithm, which takes this approach and was proposed by He et al. [13].

Cluster-based local outlier factors

In 2003, He et al. proposed the cluster-based local outlier factors (CBLOF) algorithm [13]. The algorithm has similarities to both cluster-based algorithms and the algorithms based on the idea of Local Outlier Factors (LOF) by Breunig et al. [4]. He et al. combine these ideas by first clustering the data so that those clusters can then be used to compute an anomaly score in a way similar to LOF. Any clustering algorithm can be used in the clustering step, but of course the quality of the results of the clustering algorithm can significantly influence the quality of the results of CBLOF.

Suppose we have a dataset D. We can then compute a clustering of this dataset using an arbitrary clustering algorithm that assigns every point in D to a cluster. We can now order the clusters by size, so that we have $|C_1| \geq |C_2| \geq \cdots \geq |C_k|$. Let us denote the clusters $C_1, C_2, \ldots, C_k$ where k is the number of clusters. The intersection of any pair of these clusters should be empty, while the union of all clusters should contain all the observations in the dataset.

The next step in the algorithm is to find a boundary separating the Large Clusters (LC) from the Small Clusters (SC). Let us denote the index of this boundary with b. There are two possibilities to define this boundary. The first formula to find a boundary is the following:

$$(|C_1| + |C_2| + \cdots + |C_b|) \geq |D| \cdot \alpha \tag{3.1}$$

The alternative is the following:

$$(|C_b| \,/\, |C_{b+1}|) \geq \beta \tag{3.2}$$

In these equations α and β are user defined parameters. α should be set between zero and one and represents the ratio of the data that should be covered by the LC. β should be set to a value bigger than one and defines a lower bound on the relative size between two consecutive clusters. In equation 3.1, the idea is to add the biggest clusters to the set LC until some user defined part of the data is covered. This intuitively separates the bigger clusters from the smaller ones. The idea behind equation 3.2 is that a useful location for the boundary between LC and SC could also be between two clusters that have a relatively big difference in size. More specifically, it sets a boundary between two consecutive clusters if the next cluster is β times smaller than the previous cluster. These two equations might result in different values for b. If that is the case, the smaller value is used.

After finding the boundary, b, we create the sets LC and SC. LC simply contains all clusters with an index smaller or equal to the boundary, or more formally $LC = \{C_i \mid i \leq b\}$. SC contains the remaining clusters, more formally denoted as $SC = \{C_i \mid i > b\}$.

The final step is to compute the CBLOF scores for every element in D. He et al. use the following equation to compute this score for $t \in D$.

$$CBLOF(t) = \begin{cases} |C_i| \cdot \min(dist(t, C_j)) & : t \in C_i, C_i \in SC \text{ and } C_j \in LC \\ |C_i| \cdot dist(t, C_i) & : t \in C_i, C_i \in LC \end{cases} \tag{3.3}$$

As can be seen in the equation, the first option in the piecewise function is used for the data points that belong to a cluster in SC, while the second option covers the points that are part of LC. The distances to the clusters in the equation are the distances to the representative point of the cluster. This can, for example, be the mean of the cluster, but depends on the algorithm used for clustering. The formula for points in SC gets the distance between the current point and the nearest cluster part of LC. For elements in LC, the distance to the cluster the element is part of is used. To incorporate the size of the cluster the elements belong to, the distances are multiplied with the cardinality of the cluster they belong to to get the CBLOF score.

The above definition is not very intuitive, so we will discuss how it works. The points in the large clusters (i.e. LC) are considered to be the normal points, while the points in the small clusters (i.e. SC) are considered to be possible anomalies. For every large cluster, we compute a point that represents that cluster (this can, for example, be the mean of all points in the cluster). The anomaly score of a point that is part of a large cluster is then defined as the distance between that point and the central point of the cluster. This means that the anomaly score increases when the point is further away from the point that represents the cluster. For points that are not part of a large cluster, the anomaly score is defined as the distance to the closest point that represents a large cluster. This way, the anomaly scores for points that are far away from all large clusters will be higher than the anomaly scores for the points that are still relatively close to one or more large clusters. This method to compute anomaly scores has some similarities to LOF, with the important difference that the distances are computed with clusters instead of individual points.

This algorithm was not yet part of the ELKI framework [23], but was a good candidate to be added. Therefore I implemented this algorithm myself within the ELKI framework and sent it back to the ELKI maintainers. They accepted the contribution and therefore this algorithm is now part of ELKI and available for anyone to use. The algorithm is set up so that it can use many different clustering algorithms available within ELKI. That makes it possible to easily conduct experiments with the CBLOF algorithm combined with many different clustering algorithms to find an optimal combination without any extra coding efforts required. It is also possible to use different distance functions in the computations of the CBLOF scores. This may be useful for some datasets, but is probably not required for our experiments.

3.3 Temporal approach

Contrary to the non-temporal approaches, the algorithms with a temporal approach do give a specific meaning to the time aspect of the data. A possible way to use this information can, for example, be to attach less value to older observations while more recent observations have more influence on the results. It can also be used to find, for example, a daily repeating pattern. While there may be ways to do this with non-temporal algorithms as well by, for example, extracting the right features, temporal algorithms have an advantage in this area since they were designed with the temporal idea in mind. This is basically a bit of domain specific knowledge that is used in an attempt to improve the results at the cost of the generality of the algorithm. However, this loss of generality is not a problem in our case study because all our data does have a temporal aspect, giving us the possibility to use this kind of algorithm.

We are going to discuss an algorithm based on Markov Modulated Poisson Processes that was proposed by Ihler et al. [15]. This algorithm roughly takes an evolving prediction approach. The general idea behind algorithms using the evolving prediction approach is to create and maintain a model of the data. As new data arrives, this model is updated and used to classify the new data.

The approach is based on time-varying Poisson processes. The algorithm is built to analyze data using the observed counts in fixed, consecutive time periods. This matches the aggregations we discussed in Section 2.1. Ihler et al. distinguish two kinds of anomalies in the data. The first group, which is not interesting, is the group containing all the slight deviations which are basically caused by noise in the data. The second group, which is the group of anomalies we want to find, is the group containing the actual anomalies indicating significantly different behaviour. Ihler et al. attempt to separate these two by defining a model of uncertainty which indicates how unusual a point in the data is. Additionally, they incorporate persistence in the model. This addition increases the likeliness of a point being an anomaly if it is part of a set of multiple consecutive points that are more likely to be an anomaly. This decreases the probability that a single point that is slightly deviating from the norm is marked as an anomaly.

The proposed probabilistic model uses two aspects of time. The first is the time of day and the second is the day of the week. The algorithm can optionally be configured to link the rates or time profiles of all days, weekdays and weekend days separately, or all days separately. A Bayesian approach is used to incorporate learning and inference into the model.

3.3.1 Method outline

We will discuss the method proposed by Ihler et al. in two parts. First, we take a look at the probabilistic model that they use. After that, we will discuss the way learning and inference is performed.

Probabilistic model

The algorithm assumes that the counts are determined by a combination of two functions. The first represents the normal behaviour, which only depends on the time (day of week and time of day). The second models the anomalies we want to find, which disturb the normal behaviour. This can be captured in a model as follows:

$$N(t) = N_0(t) + N_E(t)$$

In this model, $N_0(t)$ models the normal behaviour while $N_E(t)$ models the anomalous behaviour. To be able to find the deviations modeled by the second function, we first have to find the baseline represented by the first function. We will therefore start with an explanation of how the normal behaviour is modeled.

The normal behaviour is modeled using a model based on a model proposed by Scott [25]. This model is based on the Poisson distribution. However, instead of using a constant λ, a function depending on the time is used. This function is defined as

$$\lambda(t) = \lambda_0 \; \delta_{d(t)} \; \eta_{d(t),h(t)}$$

In this function, $d(t)$ represents the day of the week as a number from 1 to 7. $h(t)$ represents the time of the day as the number of the aggregation interval the observation is a part of. λ_0 is the average rate of the Poisson process over an entire week. δ_i adds a per day variation. This makes it possible to, for example, model a higher average rate for Mondays and a lower one for Sundays. Finally, $\eta_{j,i}$ models the variations during the day for a time period i on day j. This makes it possible to, for example, model a higher rate during the day and a lower rate at night. The day of the week is also taken into account here to allow different patterns during a day depending on the day of the week.

The second function we discussed above models the anomalous behaviour. To model this function, Ihler et al. use a binary process that indicates whether the data at a given time indicates the presence of an anomaly:

$$z(t) = \begin{cases} 1 & \text{if there is an event at time } t \\ 0 & \text{otherwise} \end{cases}$$

The probability distribution over $z(t)$ is modeled by a Markov process over time. The following transition probability matrix is used for this process:

$$M_z = \begin{pmatrix} 1 - z_0 & z_1 \\ z_0 & 1 - z_1 \end{pmatrix}$$

This gives us an expected time between anomalies of $1/z_0$, while each anomaly has an expected duration of $1/z_1$. Using the beta distribution, β, we give z_0 prior

$$z_0 \sim \beta(z; a_0^Z, b_0^Z)$$

and z_1 prior

$$z_1 \sim \beta(z; a_1^Z, b_1^Z).$$

Using the probability distribution, $z(t)$, we can model the second function as follows:

$$N_E(t) \sim \begin{cases} 0 & \text{if } z(t) = 0 \\ P(N; \gamma(t)) & \text{if } z(t) = 1 \end{cases}$$

In this function, P is a Poisson distribution and γ is defined as:

$$\gamma(t) \sim \Gamma(\gamma; a^E, b^E)$$

To make these functions a bit more clear, Ihler et al. also provide two graphical models that show the relations between the variables and functions. These graphical models can be found in figures

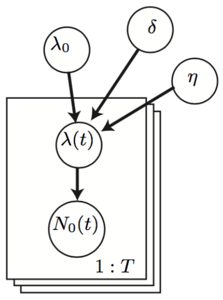

FIGURE 3.2: Graphical model for $\lambda(t)$ and $N_0(t)$. The parameters λ_0, δ, and η are the periodic components of $\lambda(t)$ and couple the distributions over time [15].

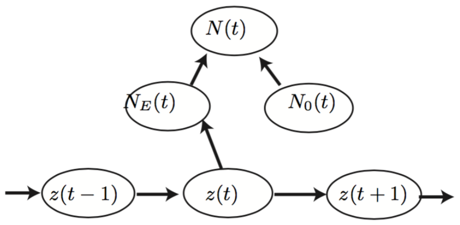

FIGURE 3.3: Graphical model for $z(t)$ and $N(t)$. The variables are coupled over time by the Markov structure of $z(t)$ [15].

For our case study, we are looking for the periods of time where $z(t)$ is 1, since this indicates that something anomalous is going on at time t. Optionally, we can also look at $\gamma(t)$ or $N_E(t)$ to find the magnitude of the deviation caused by the anomaly. This magnitude may help answer the second research question, because it can provide some extra information about the reason something is classified as an anomaly.

This probabilistic model also has a disadvantage that is mentioned by the authors themselves. This disadvantage is the fact that the model does not support negative contributions to the expected count. This means that anomalies that have a negative contribution to the number of observations cannot be detected.

Learning and inference

The second part of the method covers the learning part of the model. This part of the method is related to the third research question, which considers the handling of changes over time and influence of past anomalies.

Ihler et al. simplify this problem by first assuming that the data covers an integral number of weeks [15]. This is not a real restriction, because the difference between the available data and the next integral number of weeks can be filled as unobserved, thereby not influencing the result.

As we can see in figures 3.2 and 3.3, λ_0, δ, η, z_0, and z_1 are all conditionally independent. Therefore, we can compute the maximum a posteriori (MAP) estimates or draw posterior samples of the parameters $\lambda(t)$ and $\{z_0, z_1\}$ using the complete data $\{N_0(t), N_E(t), z(t)\}$.

Using the above information, posterior distributions can be inferred over the variables using Markov Chain Monte Carlo (MCMC) methods [9, 10]. Ihler et al. propose to do this by iterating between drawing samples of the hidden variables $\{z(t), N_0(t), N_E(t)\}$ and the parameters given the complete data. We will discuss both steps in more detail later in this section. The complexity of each iteration in the above process is linear in the length of the time series. Ihler et al. claim this process converges quite quickly.

We will now discuss the sampling of the hidden variables given parameters first. After that, we will discuss the process of sampling the parameters given the complete data.

To sample the hidden variables, we will use a variant of the forward-backward algorithm by Baum et al. [3]. Using this algorithm, we can draw a sample sequence $z(t)$ given the transition probability matrix, M, and the periodic Poisson mean, $\lambda(t)$. For each $t \in \{1 \dots T\}$, the conditional distribution, $p(z(t)|\{N(t'), t' \leq t\})$, is computed in the forward pass. To do this, the following likelihood functions are used:

$$p(N(t)|z(t)) = \begin{cases} P(N(t); \lambda(t)) & z(t) = 0 \\ \sum_i P(N(t) - i; \lambda(t)) \mathrm{NBin}(N; a^E, b^E/(1+b^E)) & z(t) = 1 \end{cases}$$

For the backward pass, samples are drawn as follows:

$$Z(T) \sim p(z(t)|z(t+1) = Z(t+1), \{N(t'), t' \leq t\})$$

We can compute $N_0(t)$ and $N_E(t)$ given $z(t) = Z(t)$. To do this, we take $N_0(t) = N(t)$ if $z(t) = 0$. If $z(t) = 1$, we draw $N_0(t)$ from the discrete distribution

$$N_0(t) \sim f(i) \propto P(N(t) - i; \lambda(t)) \mathrm{NBin}(i; a^E, b^E/(1+b^E))$$

We set $N_E(t) = N(t) - N_0(t)$. In the case of unobserved data, $N_0(t)$ and $N_E(t)$ are drawn independently given $z(t)$.

The simplification to use an integral number of weeks helps us with the sampling of the parameters given the complete data. Because of this simplification, the time period covered by our data can be broken down as follows:

$$T = 7 \cdot D \cdot W$$

In this formula, D is the number of time intervals in a day, and W is the number of weeks. Because we have an integral number of weeks, the complete data likelihood is given by:

$$\prod_t e^{-\lambda(t)} \lambda(t)^{N_0(t)} \prod_t p(Z(t)|Z(t-1)) \prod_{Z(t)=1} \mathrm{NBin}(N_E(t))$$

Because we can choose conjugate prior distributions, the posteriors are given by distributions of the same form with parameters given by sufficient statistics. To make the definition of the posterior distributions a bit easier, we define the following:

$$S_{i,j} = \sum_{t: \substack{d(t)=i, \\ h(t)=j}} N_0(t),$$

$$S_i = \sum_j S_{i,j},$$

$$S = \sum_i S_i$$

Using these definitions, we get the following posterior distributions:

$$\lambda_0 \sim \Gamma(\lambda; a^L + S, b^L + T)$$

$$\frac{1}{7}[\delta_1, \ldots, \delta_7] \sim \mathrm{Dir}(\alpha_1^d + S_1, \ldots, \alpha_7^d + S_7)$$

$$\frac{1}{D}[\eta_{j,1}, \ldots, \eta_{j,D}] \sim \mathrm{Dir}(\alpha_1^h + S_{j,1}, \ldots, \alpha_D^h + S_{j,D})$$

To sample z_0 and z_1, we do something similar. We first define the following simplification:

$$Z_{i,j} = \sum_{t:\substack{z(t)=i,\\ z(t+1)=j}} 1 \qquad \text{for } i = \{0,1\},\ j = \{0,1\}$$

We can then compute z_0 and z_1 as follows:

$$z_0 \sim \beta(z; a_0^Z + Z_{01}, b_0^Z + Z_{00})$$

$$z_1 \sim \beta(z; a_1^Z + Z_{10}, b_1^Z + Z_{11})$$

The above functions can be tuned to change the number of detected events by changing the priors to the transition parameters of $z(t)$. Note that bad values for these parameters can also cause overfitting of the data. Since the presence of anomalies is modeled by $z(t)$, we can get posterior probabilities that indicate when anomalies occur using $p(z(t)|\{N(t)\})$.

3.3.2 Implementation notes

The authors published their own implementation of the algorithm[1]. Since no ELKI or other easy to use implementation is available, we will be using this original implementation in our experiments. The authors wrote their code in Matlab, which has a few consequences. First of all, Java and Matlab cannot communicate directly, so we have to make some modifications to allow our tool to send data and retrieve the results. Another important consequence is the fact that we cannot compare the computational performance with the other algorithms we discussed. This comparison would be unfair because Matlab and Java programs may show fundamentally different performance characteristics. Additionally, this implementation may be far more or less optimized than the implementations of the other algorithms.

However, despite these disadvantages, we are still using this implementation because the algorithm is fairly complex. The implementation by Ihler et al. took roughly 300 lines of code. Besides the fact that it would take a lot of time to translate the Matlab code to Java, it is also likely that mistakes would be made in this process. This could result in invalid results in our experiments or, if the mistakes are detected, cost even more time.

[1]Algorithm implementation by Ihler et al. [15]:
http://www.ics.uci.edu/ ihler/code/event.html

Chapter 4

Experiments

We can divide our experiments into two parts. First, we will perform some experiments using artificial datasets to test some basic functionalities of the implemented algorithms. After that, we will experiment using the data we gathered from different servers at Flexyz.

4.1 Artificial data

Since the real world dataset does not contain any labels, it is more difficult to validate our results. Therefore, we use some basic artificial datasets to check whether the algorithms we use will be able to detect some typical anomalies we expect to find in our real world dataset. This is of course not a real validation or test for the algorithms, but it can help us answer the research questions and it may show us flaws that we might miss if we only worked with unlabeled data. We will also test how the algorithms handle the absence of anomalies. This part is also fairly important because not every real world case will contain anomalies either.

We will first discuss the details of the artificial data. After that we look at the setup used for the experiments and we conclude with the results we found using the artificial data.

4.1.1 Used data

The used datasets all spawn 28 days. We will experiment with datasets with different properties, including noise, wave patterns, anomalies, and different weekend patterns. The dataset will be divided into buckets of five minutes, so that there are 288 buckets per day. The data for each bucket is a simple count, indicating the number of observations in the given time interval. We use a base value of a 1000 for each bucket. The waves we mentioned above will cause a maximum deviation of 500. The influence of the other properties depends on the dataset and will be discussed below.

The datasets that were generated and that are used for this part of the experiments can be found in table 4.1. The *dataset* column contains an identifier so that we can easily refer to a dataset. In the *weekend* column, the relative ratio for weekend days is given. Noise is given in the third column and the value displayed is the λ value for a Poisson distribution. This noise is either added or subtracted. Both addition and subtraction have an equal chance of happening. In the *wave* column, the number of hours a single wave in the data is specified. A zero in that column means that the counts do not vary throughout the day. The last two columns give information about the anomalies that are present. The first of the two gives the probabilities that at any given bucket the data changes from non anomalous to anomalous and from anomalous to non anomalous respectively. Note that zeroes in that column indicate the absence of anomalies in the data. The last column contains the ratio between the anomalous data and the normal counts in case of an anomaly.

Dataset	Weekend (ratio)	Noise (λ)	Wave (hours)	Anomaly prob. (start/end)	Anomaly (ratio)
1	1	0	0	0/0	N/A
2	0.5	0	0	0/0	N/A
3	1	0	24	0/0	N/A
4	1	20	24	0/0	N/A
5	0.5	0	24	0.005/0.9	0.5
6	0.5	20	24	0.005/0.9	2
7	0.5	20	0	0.002/0.95	1.5

TABLE 4.1: Overview of the used artificial datasets.

We will discuss dataset 6 as an example. This dataset contains half the observations on weekend days, compared to other days of the week. Additionally, noise causes the counts to be deviating from the norm by either adding or subtracting samples from a Poisson distribution with a λ of 20. Furthermore, the data has a wave pattern with a period of a day. This means that there will be less than average observations during the night and more than average during the day. The probability that anomalous data is present at a bucket while the previous bucket contained normal data is 0.5%. If the last bucket was anomalous, the chance that this bucket is also anomalous is 90%. When an anomaly is present, this means that the counts are twice the normal counts at any given time.

4.1.2 Experimental setup

The approach for the experiments with the artificial data is different from the approach used with the real data. We will discuss the setup for the real data in Section 4.2.2. In Chapter 3, we discussed four algorithms and their implementations. These algorithms were KNN [21], LOF [4], CBLOF [13], and MMPP [15]. The experiments in this section will be performed using these four algorithms.

For this part of the experiments, we have labeled data. We can therefore compute both the precision and the recall for each algorithm and configuration. The number of configurations we have to test depends on the algorithm, or, more specifically, the parameters of the algorithm. Besides the parameters for the algorithms, we can also tune the threshold for outliers. We will test the following parameters for the different algorithms:

- *KNN*: $\{k \in \mathbb{N} \mid 1 \leq k \leq 10\}$
- *LOF*: $\{k \in \mathbb{N} \mid 1 \leq k \leq 10\}$

- *CBLOF*:
  - $k \in \{1, 2, 4, 5, 8, 10, 20\}$
  - $\alpha \in \{0.7, 0.8, 0.9, 0.95\}$
  - $\beta \in \{1.5, 2, 3\}$
- *MMPP*: No parameters

Besides these parameters we also tune the threshold. We use the following general formula to compute the threshold:

$$a \cdot mean + b \cdot variance$$

In this function, a and b are parameters we tune. $mean$ is the mean of the anomaly scores of the algorithm and $variance$ is the variance of the anomaly scores. We can find the values for a and b in a post processing step because it is simply a filter over the results of an algorithm. It is not part of the algorithm itself and therefore we do not need to run the algorithm again to test different values for a and b. For a we pick a value from the set $\{1, 1.5, 2\}$. The possible values for b are $\{0, 1, 1.5, 2\}$. Since the optimization can be done in a post processing step that is relatively fast compared to the algorithm execution, we do a complete grid search over the possible values for a and b. Note that we do this for every algorithm and parameter set separately, because both may significantly influence the anomaly scores. It can therefore give better results if we optimize the threshold for each specific case. The threshold is the same for every dataset however.

To find the optimal parameter values for the algorithm specific parameters, we do need to run the algorithm for every combination of parameter values we want to test. Because the number of algorithms and the possible number of parameter combinations is relatively limited, we also do a complete grid search to find the optimal algorithm specific parameter values.

Now that we have determined which algorithm configurations we need to run, we also have to specify a way to make a fair comparison between the results to determine which algorithm and which configuration performs best. First of all, we need to make sure the results are realistic and not caused by overfitting. To do this, we split the datasets into a training and test part. As we mentioned in Section 4.1.1, all datasets contain data for four weeks. Because we are dealing with temporal data, we cannot split the data into arbitrary parts and we can therefore not do k-fold cross validation for an arbitrary k. Note that we cannot do this because this might make it impossible to detect certain trends over time in the data. We will therefore split the data into two parts. The first three weeks will be used for training and the last week will be used for testing.

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

TABLE 4.2: Example confusion matrix indicating the meaning of TP, FP, TN, and FN.

The last thing we need is a way to assign a score to the results, so that we can compare them. We will use several concepts explained by Rijsbergen for this part [22]. We can extract a confusion matrix with the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) counts from the results of an algorithm. See Table 4.2 for an explanation of the exact definitions of these counts. Next, we can compute the precision and recall as follows:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Since optimizing one of those measures is trivial, we need a way to combine them to get a meaningful score. We use the F_1 score to do this. The F_1 score can be computed as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

This harmonic mean of precision and recall weights precision and recall equally, but penalizes significantly different scores, thereby preventing optimization of only one of the factors. Using the above, we get an F_1 score for every dataset. We combine these scores into a single score for an algorithm by computing the average over all datasets.

The complete procedure used for this part of the experiments can be found in Listing 3. The pseudocode presented there represents the actual implementation of the above explanation.

There is one pre-processing step that we have not yet discussed. This step applies only to the ELKI algorithms (KNN, LOF, CBLOF). Because these algorithms do not give any special meaning to the time related features, we perform a feature wise normalization. This step is necessary because the counts in a log bucket can be very big (i.e. many thousands or even more). The big variety in this feature can overshadow the variation in the other features, which can make it difficult to actually use the information given by those features. To prevent this we scale all the features to an interval from zero to one, mapping the minimum and maximum value for each feature to zero and one respectively. This step is not used for the temporal algorithm because this algorithm uses the time related features differently. The only generic feature that remains is therefore the count for a bucket. Since this feature is not put in a feature space with any other features, scaling does not influence the results. We will also run the experiments without this pre-processing step to see how it affects the results.

4.1.3 Results

The summarized results can be found in Table 4.3. More detailed results of the experiments with the artificial data can be found in Appendix A.1. Confusion matrices per dataset and per algorithm can be found in these more detailed results. We can see some big differences between the algorithms in Table 4.3. Based on those results, we can see that the KNN algorithm has the best performance overall, while the LOF algorithm comes in a close second.

Algorithm 3: Pseudocode for execution of experiments with artificial data.

```
$output \leftarrow []$
for $Algorithm\ a \in algorithms$ do
    $scores \leftarrow []$
    for $Dataset\ S \in datasets$ do
        for $Configuration\ c \in a.configurations$ do
            $results \leftarrow$ RunAlgorithm ($a$, $S.train$, $c$)
            $mean \leftarrow$ GetMean ($results$)
            $stdev \leftarrow$ GetStdev ($results$)
            for $a, b \in thresholds$ do
                $threshold \leftarrow a \cdot mean + b \cdot stdev$
                $precision, recall \leftarrow$ Evaluate ($results$, $threshold$)
                $F_1 \leftarrow 2 \cdot precision \cdot recall/(precision + recall)$
                Add ($c$, $a$, $b$, $F_1$) to $scores$
            end
        end
    end
    $optimal \leftarrow$ GetParametersWithBestScore ($scores$)
    $testScores \leftarrow []$
    for $Dataset\ S \in datsets$ do
        $results \leftarrow$ RunAlgorithm ($a$, $S$, $optimal$)
        $testResults \leftarrow results.test$
        $threshold \leftarrow$
          $optimal.a \cdot$ GetMean($results$) $+ optimal.b \cdot$ GetStdev($results$)
        $precision, recall \leftarrow$ Evaluate ($testResults$, $threshold$)
        $F_1 \leftarrow 2 \cdot precision \cdot recall/(precision + recall)$
        Add ($optimal$, $F_1$) to $testScores$
    end
    $totalScore \leftarrow \sum testScores/7$
    Add row with $a$, $totalScore$, $optimal$, and $testScores$ to $output$
end
return $output$
```

Algorithm	KNN	LOF	CBLOF	MMPP
Score	0.781	0.755	0.571	0.161

Parameters	k = 4
	a = 1.5
	b = 0
		k = 6
		a = 1.5
		b = 0
			α = 0.7
			β = 1.5
			k = 8
			a = 1
			b = 1
				a = 2
				b = 0

1	1.000	1.000	1.000	0.000
2	1.000	1.000	1.000	0.000
3	1.000	1.000	1.000	0.000
4	0.000	1.000	1.000	0.000
5	0.469	0.410	0.000	0.000
6	1.000	0.449	0.000	0.616
7	1.000	0.429	0.000	0.508

TABLE 4.3: F_1-scores of implemented algorithms on the artificial datasets with normalization.

Another interesting observation we can make is the fact that the scores for datasets 1 through 4 are always either 0 or 1. This can be easily explained however. The score that is shown is an F_1 score which is a combination of precision and recall as we saw in the previous section. Since there are no actual anomalies in datasets 1 through 4, the precision can either be 1 (no anomalies reported) or 0 (one or more anomalies reported). The recall is always 1 since we cannot miss any anomalies. This results in an F_1 score of 0 if an anomaly is reported and an F_1 score of 1 if no anomalies are reported. Based on the results, it seems that the MMPP algorithm cannot handle the absence of anomalies very well. Furthermore, the limitation of the MMPP algorithm we discussed in Section 3.3.1 (i.e. anomalies with a negative contribution to the observation count cannot be detected) shows itself in the score for dataset 5. This is the only dataset containing anomalies with a negative contribution and the algorithm gets a rather bad score here, while it performs relatively well on the datasets containing anomalies with a positive contribution to the count.

We can gain some more insight into some of the algorithms if we look at the more detailed results in Appendix A.1. We can for example see that the CBLOF algorithm does not predict any anomalies at all. Furthermore it seems that the LOF and KNN algorithms are also rather conservative in marking data points as an anomaly. The LOF algorithm does not have any false positives while the KNN algorithm has only one. However, both still find quite a lot of true positives. Especially the KNN algorithms still has a high number of true positives, which of course also explains its generally good score. Another number that stands out is the high number of false positives produced by the MMPP algorithm. Although this algorithm manages to find a lot of actual anomalies, this comes at the cost of a high number of false positives.

The results for the different algorithms without the normalization preprocessing step can be found in Table 4.4. The results for the MMPP algorithm are, of course, the same since the normalization step was not used for this algorithm. For the other algorithms we find some surprising results however. While the KNN and LOF algorithm had better scores while using normalization, the CBLOF algorithm works a lot better without normalization. Its score actually comes close to the score of the KNN algorithm with normalization.

Algorithm	KNN	LOF	CBLOF	MMPP
Score	0.622	0.59	0.766	0.161

Parameters	k = 4
	a = 1.5
	b = 1.5
		k = 4
		a = 1.5
		b = 1.5
			α = 0.9
			β = 3
			k = 1
			a = 2
			b = 0
				a = 2
				b = 0

1	1.000	1.000	1.000	0.000
2	1.000	1.000	1.000	0.000
3	1.000	1.000	1.000	0.000
4	0.000	0.000	1.000	0.000
5	0.735	0.808	0.000	0.000
6	0.092	0.250	0.520	0.616
7	0.528	0.074	0.842	0.508

TABLE 4.4: F_1-scores of implemented algorithms on the artificial datasets without normalization.

4.2 Real world data

Contrary to the artificial datasets, we do not have any labels for the real world data. This means we will have to take a slightly different approach to validate our results. To do this, we will first discuss what data is available and what data we are using. Next, we will discuss the experimental setup for the real world data and compare it to the setup for the artificial data. Finally we will take a look at the results and make a quick comparison with the results for the artificial datasets.

4.2.1 Used data

As we already discussed in Section 2.1, a lot of data has been gathered from various production servers at Flexyz. However, a lot has changed in the applications and servers since we started gathering the logs. Therefore, not all data is usable. Some applications have, for example, only logged data for a short time which means we cannot find any meaningful trends in this data. Aside from these reasons to exclude applications from our analysis, we also do not use data from applications that are not used in production.

We do this because development, test, and acceptance environments have highly varying log statistics. If someone worked on an application for a day, an application may log millions of messages while very few messages are logged when nobody worked on that application for a day. Although variations over time also exist in applications in production environments, these changes are generally less extreme. Another reason to choose these applications is because of the simple fact that these applications are most interesting from a business perspective.

Aside from picking applications, we also have to choose a time interval for the data we use. We have roughly two months of data available. However, for the biggest part of April, the number of applications sending their log data to our infrastructure was fairly limited. Furthermore, a fairly big interruption in the process of collecting log data caused a gap at the start of June. We will therefore use four weeks of consecutive data gathered in May. Specifically, we will use the data gathered from the 1st of May until the 28th of May.

Based on the above criteria, three applications were selected. However, this does not mean that we can only perform our analysis on three datasets. These three applications generated nearly 600 unique messages. We can therefore perform experiments with different levels of aggregations of the data. We can, for example, analyze the statistics for individual messages, messages aggregated per class or messages aggregated per application. This will be discussed in more detail in the explanation of the experimental setup. The three applications that we selected logged almost 1.4 billion messages in the four weeks we are looking at, so enough observations are available to analyze the data at different levels of detail.

Some figures containing the aggregated counts can be found in Appendix B. Figure B.1 contains the aggregations for application 1. Application 2 can be found in Figure B.2 and application 3 can be found in Figure B.3. The aggregations in these graphs are per 12 hours, so a lot of patterns and deviations may not be visible at this level of detail. We can already see some patterns however. For example, in all datasets we can see that there is a fairly big difference between the number of log records before noon and after noon. Furthermore we can see in dataset 1 that the number of records that is logged is a lot less on every Sunday. In application 2 and especially in application 3 we can see that the number of log records is quite a bit higher at the beginning of the period compared to the remaining weeks. It is interesting to see how the anomaly detection algorithms handle this.

4.2.2 Experimental setup

The same algorithms we used for the artificial data will be used with the real world data. The parameter values may be different however since we will be tuning them again with only real world data. What makes these experiments more difficult and less efficient is the fact that we do not have labeled real world data available. We will therefore have to verify the reported anomalies manually.

This also means that we cannot compute an exact recall or precision since it is not feasible to manually go through all the data to find every anomaly. Although an exact precision and recall are not possible, we can still compute them using the partial classifications. When we find an anomaly using one algorithm, but a second algorithm does not find this anomaly, we know that an anomaly was missed by that second algorithm. Since we cannot be sure that the union of all detected anomalies contains all anomalies, we cannot compute a complete recall. We can, however, still use this partial measure as a rough indicator to compare the different algorithms. A similar problem exists for the calculation of the precision. We will now discuss these points in some more detail.

Manual classification

The manual classification is an important part of these experiments because it greatly influences the results. It also takes quite some time, because a lot of possible anomalies have to be checked manually to determine whether they actually are anomalous. In the artificial results, we saw that the MMPP algorithm reported a very high number of false positives. Since we have to check possible anomalies manually, we will not be checking every reported anomaly because this is simply not possible. To speed up this process, some automation has also been done. We will now discuss how this process works.

A script automatically fetches the most likely anomalies for each algorithm configuration and dataset. Next, it fetches the data points that are likely related to the possible anomaly currently being classified. The following points were used as the related data points:

- All points on the same time and day of the week, but in different weeks than the current observation.
- Observations at the same time on the previous two and next two days.
- Points in the bucket before and after the current observation.

All these points are then presented in chronological order, together with the observation being classified. It is then manually entered whether the current observation is an anomaly or not. This classification is, amongst other things, based on the mean and variation amongst the other points, compared to the current observation. Generally, the related points in a different week are more important than the related points on a different day and the points on a different day are more important than the points at a different time. Of course, this process is influenced by a certain amount of subjectivity. If this was not the case, we would not require the algorithms discussed here, because the problem would already be solved.

We already mentioned that the process presents the possible anomalies per algorithm configuration and dataset. Since we do not validate all the data points, we need a condition to stop classifying the current dataset. When classifying the test data, we stop when the majority of the last 20 points we checked is not actually anomalous. For the training data, we use the same condition, but with the addition that we check at most 100 possible anomalies per algorithm configuration and dataset. We add this extra condition because many more configurations need to be checked for the training data and the other condition may be insufficient to keep the extent of the required manual work within feasible limits.

Data point	Actual	Manual classification	Prediction
1	Y	Y	Y
2	N	N	N
3	Y	Y	N
4	N	N	Y
5	Y	—	Y

TABLE 4.5: Example table for precision and recall computation

Of course we will go through the list of reported anomalies starting at the most likely anomaly, working our way down to reports with a lower certainty. This way the chance is smaller that a lot of actual anomalies are missed after we stopped searching. Anomalies that we found by hand will of course be saved so that we do not have to validate them again if another algorithm or configuration reports the same anomaly. This last point also means that it is likely that more manual classification results are available for an algorithm configuration and dataset than we verified for that specific configuration and dataset (i.e. the classifications from other algorithm configurations, which may cover different data points, are also available when validating the current configuration and dataset).

Comparison to experiments using artificial data

Aside from being time consuming, the manual classification we just discussed has another consequence. We already briefly mentioned it, but it affects the computation of the precision and recall. This causes an important difference compared to the results for the artificial data.

An exact computation is not possible because we have not manually classified all the data points, so we cannot be certain we know all the points that are anomalous. We therefore compute the precision and recall the same way as in the artificial experiments, with the added assumption that the points that we did not manually classify are not anomalous. We will demonstrate this using a small example.

The data for our example can be found in Table 4.5. The first column contains the numbers we use to refer to the data points. The second column indicates whether the point is actually an anomaly. The third and fourth column contain our manual classification and the prediction respectively. A Y means that the point is anomalous according to that column, while N means it is not. The "—" means that a point has not been manually classified. Note that point 5 would have been classified in our real data, but we did not classify it here, so that the example data can be kept as small as possible.

We will now demonstrate how the missing manual classification influences the results using the example above. The actual precision is $\frac{2}{3}$, because point 1 and 5 are correctly predicted to be an anomaly while point 4 is not. However, the precision we would predict for our experiments is $\frac{1}{3}$

because point 5 is assumed to be non anomalous, because this point was not classified.

The influence on the recall is slightly different. The actual recall would be $\frac{2}{3}$, because points 1, 3, and 5 are anomalous, but only 1 and 5 are predicted. However, based on the manual classifications, the recall is $\frac{1}{2}$ because point 5 is assumed to be non anomalous.

So, in this example, both the precision and recall based on the manual classification are lower than the actual precision and recall. If data point 5 would be non anomalous, the results would be the same for the manual classification and the actual results. However, if the prediction for data point 5 would be non anomalous, while it actually was anomalous, the precision and recall would be overestimated.

Based on the above example, we can see that both the precision and recall based on the manual classification results can be different from the actual precision and recall. This can result in both overestimation and underestimation. However, our assumption is that the differences caused by the incomplete manual classification are small. This is assumption is the result of the way we do our manual classification.

For every dataset, we manually classified anomalies predicted by every algorithm starting with the most likely anomalies until a majority of the last 20 was actually not anomalous. The idea behind this method is that we catch as many anomalies as possible with a limited amount of work. Even though some algorithms may not find many anomalies, algorithms that perform well find a lot of the anomalies and these results are also used for the other algorithms as we already described. Therefore, we make the assumption that the remaining data points, which have not been manually classified, do not contain many anomalies. Assuming this is true, our estimated precision and recall are not much different from the actual precision and recall, because we saw in the example that differences are caused by missing classifications of actual anomalies.

Another source of differences in the estimated prediction is caused by predicting a data point is anomalous when it is not manually classified. This should not be a significant problem because this can only be the case if the algorithm already predicted many invalid anomalies, otherwise the point would have been manually checked. Additionally, this problem should be limited due to the used threshold (i.e. it is never optimal to predict anomalies amongst the unclassified data points, therefore an optimal threshold should exclude these predictions).

Although the precision and recall are possibly not completely accurate, we will still be using them to validate the results, because there is no better method available. This does mean that the scores for this part of the experiments cannot be directly compared to the results for the artificial data or the results in other research. However, it does give us the ability to compare the achievements of the different algorithms and configurations we used.

Aside from these important differences in validating the results, we can also do a lot of things the same way as we did for the artificial data. We will use the same possible parameter configurations and threshold function. The experiments will also be performed both with and without normalization for the ELKI algorithms. The procedure for this part of the experiments therefore still has a lot of similarities to the experiments for the artificial data.

Use of data

Instead of the 7 datasets used for the artificial data, we will be using 18 datasets for this part of the experiments. The aggregations in these datasets can be grouped in different levels of detail. We will use 9 datasets that contain aggregations for a single unique message. Furthermore 6 datasets will contain the aggregations for all messages within a single class. In this case, a class is a Java class and the aggregation therefore includes all messages that are logged from within that class. The other 3 datasets will contain aggregations over the entire application. This results in the following contents for our datasets:

- Application 1:
  - *Messages*: datasets 1, 2, 3
  - *Classes*: datasets 10, 11
  - *Application*: dataset 16
- Application 2:
  - *Messages*: datasets 4, 5, 6
  - *Classes*: datasets 12, 13
  - *Application*: dataset 17
- Application 3:
  - *Messages*: datasets 7, 8, 9
  - *Classes*: datasets 14, 15
  - *Application*: dataset 18

The given dataset numbers match the numbers in the results. For example, when we discuss datasets 2 and 11 in the results, we can use the above list to find out that both are part of application 1. We can also see that dataset 2 contains aggregations for a single message, while dataset 11 contains aggregations for all messages within a Java class. The messages and classes are randomly selected within the applications.

4.2.3 Results

The summarized results for the real world data can be found in Table 4.6. More detailed results, including confusion matrices over all algorithms and datasets, can be found in Appendix A.2. Before we discuss the results in more detail, there are a few things that should be taken into account with respect to the interpretation of the results. We already mentioned the first and most important one in the explanation of the experimental setup. The recalls and precisions that were used to compute the F_1 scores are not necessarily correct and these scores can therefore not easily be compared to results of other experiments. It should also be taken into account that there is a certain level of subjectiveness to the manual classification process, which may influence the results. This is caused by the fact that there are no strict rules that determine when something is an anomaly.

Based on the results in Table 4.6, we can already make quite a few interesting observations. First of all, the KNN algorithm seems to give the best performance of the four algorithms, just like in the artificial experiments. As we could expect from the results for the artificial data, the performance for the KNN and LOF algorithm was better with normalization, while the CBLOF algorithm performed better without it. The optimal parameter values are quite different from the parameter values for the artificial data though.

When we look at the results for the individual datasets, we can see that the best score is always achieved by either the KNN or the LOF algorithm. For dataset 9, we see the same kind of scores we saw in some of the artificial datasets. This dataset almost exclusively consists of the same constant value. The few values in the test part of the dataset that were different were not reported in the list of most extreme anomalies by any of the algorithms and were therefore not marked as anomalies. Since the KNN algorithm was the only one that did not report any anomalies for this dataset, this resulted in a perfect score for that algorithm while the other algorithms got a score of 0.

Looking at the results for the individual datasets, we can furthermore see big differences in the scores for the different datasets. This may simply be explained by big differences between the different datasets. During manual validation, it turned out that there does not seem to be much of a pattern in some of the datasets. This results in a dataset where it is nearly impossible to find some sort of normal behaviour, which in turn causes many false positives and false negatives.

Using Table 4.6, we can also determine whether the different levels of aggregations we used have an impact on the performance. The comparison would of course be more reliable if we had more datasets, but with the data we used in the experiments, it seems that the analysis at the level of individual messages works best. For the best performing algorithm (i.e. KNN), this level has an average score of 0.455 while the average score at the class level is 0.379 and the average score at the application level is 0.109. Especially the gap between the application level and the other levels is relatively big. Although the presented F_1 scores are probably not normally distributed, we will give the standard deviations for each level to give an indication of how different the scores within each level are. For the message level, the standard deviation is 0.346. It is 0.320 for the class level and 0.115 for the application level.

If we look at the confusion matrices in Appendix A.2, we find some unexpected numbers. While the MMPP algorithm was the only algorithm that reported a lot of false positives for the artificial data, all algorithms show this behaviour with the real world data. Although this might partially be explained by anomalies that were not manually checked due to the conditions discussed in the experimental setup, it is unlikely that this explains a majority of the false positives. If we look at the confusion matrices per dataset, we also see that this behaviour is not caused by any specific dataset.

Algorithm	KNN	LOF	CBLOF	MMPP
Score	0.372	0.242	0.164	0.116

Parameters	k = 1
	normalized
	a = 1.0
	b = 1.5
		k = 9
		normalized
		a = 1.0
		b = 1.0
			k = 2
			α = 0.8
			β = 2.0
			not normalized
			a = 1.0
			b = 0.0
				a = 1.0
				b = 2.0

Messages				
1	0.285	0.383	0.234	0.172
2	1.000	0.357	0.333	0.250
3	0.462	0.154	0.133	0.190
4	0.557	0.667	0.370	0.185
5	0.266	0.462	0.234	0.175
6	0.357	0.590	0.279	0.203
7	0.048	0.035	0.024	0.019
8	0.117	0.152	0.094	0.045
9	1.000	0.000	0.000	0.000
Classes				
10	0.630	0.088	0.044	0.054
11	0.545	0.414	0.359	0.320
12	0.150	0.133	0.077	0.065
13	0.802	0.476	0.444	0.210
14	0.048	0.045	0.024	0.013
15	0.097	0.114	0.075	0.037
Applications				
16	0.241	0.196	0.173	0.106
17	0.051	0.055	0.031	0.023
18	0.034	0.036	0.015	0.029

TABLE 4.6: F_1-scores of implemented algorithms on the real world datasets.

Chapter 5

Evaluation

We already looked at the results in a bit more detail in the previous chapter. In this chapter, we will look at the results from a different perspective. We will therefore discuss what these results mean for both the scientific and the business perspective we discussed in Chapter 1. The research questions will be used to evaluate the results from a scientific perspective. To evaluate the results from a business perspective, we will look at the solutions provided to the requirements we discussed in Section 1.1.1.

5.1 Scientific evaluation

The following research questions were posed in the introduction:

(1) *Can on-line anomaly detection be performed on log data while limiting the number of false positives?*

(2) *Can the anomalies be placed in context so that we can clearly state why they are classified as anomalies?*

(3) *Can the anomaly detection model handle changes over time without future predictions being influenced by incidental anomalies?*

The answers to these questions cannot entirely be inferred from the results presented in the previous chapter, because not all underlying data was presented. This is not possible due to the size of this underlying data. We will, however, discuss the relevant details here to be able to give answers to the questions. Furthermore, the answers to the questions, of course, differ depending on the used algorithm.

5.1.1 Research question 1

The first question consists of two parts. The on-line part depends partially on the infrastructure and partially on the analysis process. The infrastructure used in this project can get the log records from the source application to the central point where analysis happens in a matter of seconds. However, working with aggregations is a core part of the approach we took and these aggregations consist of buckets that aggregate logs over a period of 5 minutes. This means that we can do a new analysis every 5 minutes. This is not completely real-time but still fast enough for the intended purposes.

The last important step is the time it takes for the anomaly detection algorithm to compute the anomaly scores. This varied per algorithm and even per dataset in our experiments. While this took less than a second for the ELKI algorithms, the MMPP algorithm took nearly 90 minutes on the most difficult dataset, while the other datasets took from anywhere between a few seconds and a few minutes. As we mentioned before, a direct comparison in computational performance cannot be made due to the implementation differences, but unless an optimized version of the MMPP algorithm is a few orders of magnitude faster, it may not be usable for on-line analysis. The ELKI algorithms seem to be usable for this purpose though.

The second part, limiting the number of false positives, has a less positive answer if we look at the results from the previous chapter. Although the answer varies a bit across the different algorithms, there are generally a lot of false positives. If we look at the results in Appendix A.2, we see that all the algorithms have at least 8 times more false positives than true positives. The actual numbers may, as we discussed in the previous chapter, be somewhat different, but this part does not look very positive. One way to solve this problem would be to simply increase the threshold. This, however, comes at the cost of an increase in false negatives. It is a business motivated decision to decide what is an optimal balance. A new optimal configuration can then be computed by replacing the F_1 score by a version that emphasizes the more important part.

Although the answer to the question is a bit more complicated than a simple yes or no, the answer based on our experiments is no because the number of false positives is simply too high. However, this does not necessarily mean that the algorithms are not usable as we discussed above.

5.1.2 Research question 2

The answer to the second research question can be split into two cases. In the introduction we mentioned four reasons why an observation could be an anomaly. One case is formed by the observations we never saw before and the observations that are completely new. The second case is formed by the other two reasons, namely the number of occurrences increased or decreased.

The anomalies in the first case are fairly easy to detect, so the answer for that part of the question is yes. The second case is a bit more difficult. None of the algorithms give a reason why something is marked as an anomaly. Instead, they only give an anomaly score. We cannot directly infer the reason from this score. However, based on the features we used for the algorithms, we can determine to which other points the observations were likely compared. We can then compute whether the anomaly has a lower or higher number of occurrences compared to the other data points. This, in most cases, gives us a slightly more detailed reason why something is classified as an anomaly. Additionally, the related data points could of course also be shown to the user.

If a simple message stating that an observation deviates from the norm is detailed enough, the above is of course not necessary as we can distinguish between the two cases of anomalies we can encounter. With the above option taken into account, it seems possible to place anomalies in a context so that it becomes clear why something is classified as an anomaly.

5.1.3 Research question 3

The experiments have not provided much help in answering the last research question. Because it is difficult to predict what kind of trend changes we can expect in the real world data, it is not feasible to try and answer this question with the artificial experiments. However, answering this question with the real world data is also quite difficult. There are two fundamental reasons that cause this.

First of all, we have not collected data over a period of time that is long enough to detect these kinds of changes. When determining the research questions, we hoped to gather enough data to answer this question, but unfortunately not enough usable data was gathered. The second reason that makes answering this question more difficult is the fact that it would be very difficult to find these patterns. Since we do not know beforehand that these behavioural changes are present or where they occur, we will have to find them manually in order to determine how the different algorithms cope with these changes. Due to the size of the data, it can be very challenging to find these patterns.

Although we cannot answer this research question based on our own findings, it seems reasonable that subtle changes in the behaviour of the application would not cause problems with our sliding window idea. This idea is supported by a topic found in other research which is called concept drift [27]. Since we only look at the last four weeks, it can be expected that the changes would slowly be incorporated into the model. The length of this period can of course be changed if this is required. Based on the above, we might expect this research question to be answered positively, but more research is required to actually prove this for our application.

5.2 Business evaluation

We discussed the motivation for this project from a business perspective in Section 1.1.1. Some of these motivations have overlap with the scientific motivation. The biggest and most important part is of course the detection of anomalies. The answer for that part roughly matches our answer to the first research question. The analysis is possible, but it is probably desirable to modify the parameter values to reduce the number of false positives.

There was an additional business motivation however. A system that uses automatic analysis can also possibly be used to perform automatic pro-active analysis. This also has some overlap with the on-line aspect of the first research question, but requires some extra features. This is all possible though.

We already saw that the near real-time analysis is possible with, at least, the ELKI algorithms. This can then be turned into a pro-active system fairly easily. To gather all the data, we already collected all the logs and aggregated that in a central system. Our analysis tool can already use this data. An interface to show reported anomalies has also been created. The only thing that still needs to be done to achieve the required analysis is automatically running the analysis algorithm every 5 minutes. Since this is not a difficult change, this part of the business requirements is met.

Chapter 6

Conclusion

In this thesis, we have made a comparison between four algorithms with a different approach to anomaly detection. We evaluated the performance of these algorithms when applied to an anomaly detection problem for application log data analysis, which is an unsupervised learning problem in our case. The KNN, LOF, CBLOF and MMPP algorithms were all compared for this domain and evaluated using both artificial and real world data. It turned out that the relatively simple KNN algorithm gave the best performance. However, a lot of false positives were still generated using the parameter values that were optimal with respect to the F_1 score.

During this thesis project, contributions from both a scientific and business perspective were made. The scientific contributions were the following:

- Presented a comparison of different algorithms in an application domain that is relatively unexplored.
- Implemented and shared the CBLOF algorithm in the ELKI framework, so that it can easily be used in other research.
- Implemented and shared some bug fixes and other minor improvements for the ELKI framework, so that future developments become slightly easier.

The contributions from a business perspective were the following:

- Implemented a scalable infrastructure that gathers log data from applications across the network so that this data becomes searchable and visible.
- Selected and implemented some of the core components required for automated log data analysis.

6.1 Discussion

The F_1 score was used in this research as a measure for the anomaly detection performance. However, a version of this measure that focuses more on precision than recall might be more desirable because the optimal parameter values for the F_1 measure generate too many false positives. Although it may be difficult to determine what measure is ideal from a business perspective, it is fairly easy to incorporate in this method. Before an automatic analysis tool is useful for Flexyz, this still requires some discussion and possibly some experimenting.

6.2 Future work

A lot of work has already been done in this thesis, but, of course, more work can always be done. We will discuss some useful extensions here, but this list is by no means complete.

In the thesis, the manual validation of the data cost a lot of time. It is likely that same problems exist for future work in this area and it would therefore be a very useful project to create an open and freely available labeled dataset with application log data. This could significantly speed up future research, while also making it possible to compare different studies and algorithms.

Another obvious but useful extension to this work would be to extend the experiments to more algorithms to see if there are algorithms available that work even better. It would also be interesting to see how different intervals for the aggregations affect the results. Repeating the research done here, but with more data could also be useful since that could add the possibility of researching changing trends in the data and how the different algorithms cope with these changes.

Looking for generic ways to reduce the relative number of false positives could also be an interesting research area. Multivariate analysis could be an approach to do this. While we only experimented with univariate analysis, multivariate analysis looks at combinations of log messages throughout time to determine whether anomalies occur. These combinations of messages may provide more information than the individual messages. This can therefore be used to improve the quality of the results. However, the method used to combine messages is important, but also difficult. While some combinations may provide informations, many combinations may not be useful. So, although this kind of analysis may improve the results, it can be difficult to get right.

Finally there is also some work that can still be done from a business perspective. As we discussed above there are still a few things that need to be done to get this automatic analysis running in a production environment. It would definitely be interesting to see how the parts of that system that were provided in this thesis project work together. Of course it would then also be very useful to know how satisfied the users are with the automatic analysis and the results that it produces.

Appendix A

Results

A.1 Artificial data

A.1.1 Confusion matrices by algorithm

		Prediction	
		Positive	Negative
Actual	Positive	197	43
	Negative	1	13871

TABLE A.1: Confusion matrix for KNN anomaly algorithm over all artificial datasets.

		Prediction	
		Positive	Negative
Actual	Positive	67	173
	Negative	0	13872

TABLE A.2: Confusion matrix for LOF algorithm over all artificial datasets.

		Prediction	
		Positive	Negative
Actual	Positive	0	240
	Negative	0	13872

TABLE A.3: Confusion matrix for CBLOF algorithm over all artificial datasets.

		Prediction	
		Positive	Negative
Actual	Positive	147	93
	Negative	1866	12006

TABLE A.4: Confusion matrix for MMPP algorithm over all artificial datasets.

A.1.2 Confusion matrices by dataset

		Prediction	
		Positive	Negative
Actual	Positive	0	0
	Negative	355	7709

TABLE A.5: Confusion matrix for artificial dataset 1 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	0	0
	Negative	345	7719

TABLE A.6: Confusion matrix for artificial dataset 2 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	0	0
	Negative	388	7676

TABLE A.7: Confusion matrix for artificial dataset 3 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	0	0
	Negative	202	7862

TABLE A.8: Confusion matrix for artificial dataset 4 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	35	213
	Negative	402	7414

TABLE A.9: Confusion matrix for artificial dataset 5 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	301	279
	Negative	111	7373

TABLE A.10: Confusion matrix for artificial dataset 6 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	75	57
	Negative	64	7868

TABLE A.11: Confusion matrix for artificial dataset 7 over all algorithms.

A.2 Real world data

A.2.1 Confusion matrices by algorithm

		Prediction	
		Positive	Negative
Actual	Positive	678	60
	Negative	6427	15255

TABLE A.12: Confusion matrix for KNN algorithm over all real world datasets.

		Prediction	
		Positive	Negative
Actual	Positive	557	181
	Negative	5480	16202

TABLE A.13: Confusion matrix for LOF algorithm over all real world datasets.

		Prediction	
		Positive	Negative
Actual	Positive	481	257
	Negative	9535	12147

TABLE A.14: Confusion matrix for CBLOF algorithm over all real world datasets.

		Prediction	
		Positive	Negative
Actual	Positive	268	470
	Negative	6600	15082

TABLE A.15: Confusion matrix for MMPP algorithm over all real world datasets.

A.2.2 Confusion matrices by dataset

		Prediction	
		Positive	Negative
Actual	Positive	287	73
	Negative	1519	3169

TABLE A.16: Confusion matrix for real world dataset 1 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	16	8
	Negative	34	86

TABLE A.17: Confusion matrix for real world dataset 2 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	12	16
	Negative	63	113

TABLE A.18: Confusion matrix for real world dataset 3 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	43	25
	Negative	91	201

TABLE A.19: Confusion matrix for real world dataset 4 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	206	90
	Negative	1014	1614

TABLE A.20: Confusion matrix for real world dataset 5 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	300	160
	Negative	1002	1650

TABLE A.21: Confusion matrix for real world dataset 6 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	46	26
	Negative	2893	5067

TABLE A.22: Confusion matrix for real world dataset 7 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	123	57
	Negative	2100	3532

TABLE A.23: Confusion matrix for real world dataset 8 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	0	0
	Negative	2012	5992

TABLE A.24: Confusion matrix for real world dataset 9 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	74	58
	Negative	1682	6250

TABLE A.25: Confusion matrix for real world dataset 10 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	26	14
	Negative	60	140

TABLE A.26: Confusion matrix for real world dataset 11 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	151	81
	Negative	2511	5217

TABLE A.27: Confusion matrix for real world dataset 12 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	199	121
	Negative	341	1055

TABLE A.28: Confusion matrix for real world dataset 13 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	45	23
	Negative	2764	5196

TABLE A.29: Confusion matrix for real world dataset 14 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	100	44
	Negative	2181	3515

TABLE A.30: Confusion matrix for real world dataset 15 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	260	132
	Negative	2211	5461

TABLE A.31: Confusion matrix for real world dataset 16 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	55	21
	Negative	2623	5365

TABLE A.32: Confusion matrix for real world dataset 17 over all algorithms.

		Prediction	
		Positive	Negative
Actual	Positive	41	19
	Negative	2941	5063

TABLE A.33: Confusion matrix for real world dataset 18 over all algorithms.

Appendix B

Datasets

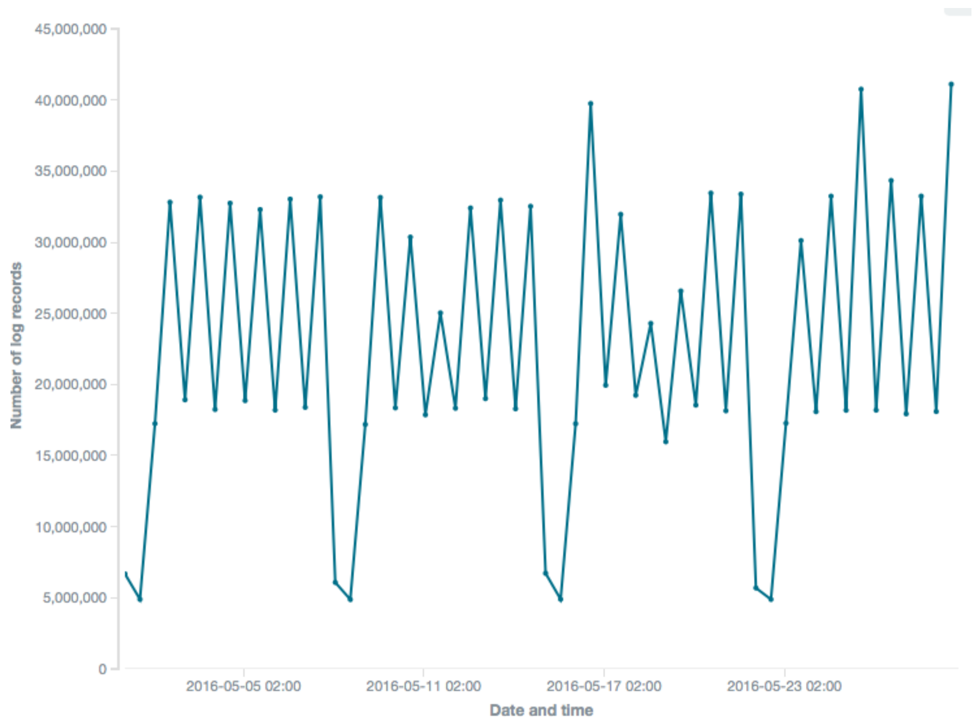

FIGURE B.1: Number of log records for application 1 aggregated per 12 hours from May 1st until May 28th.

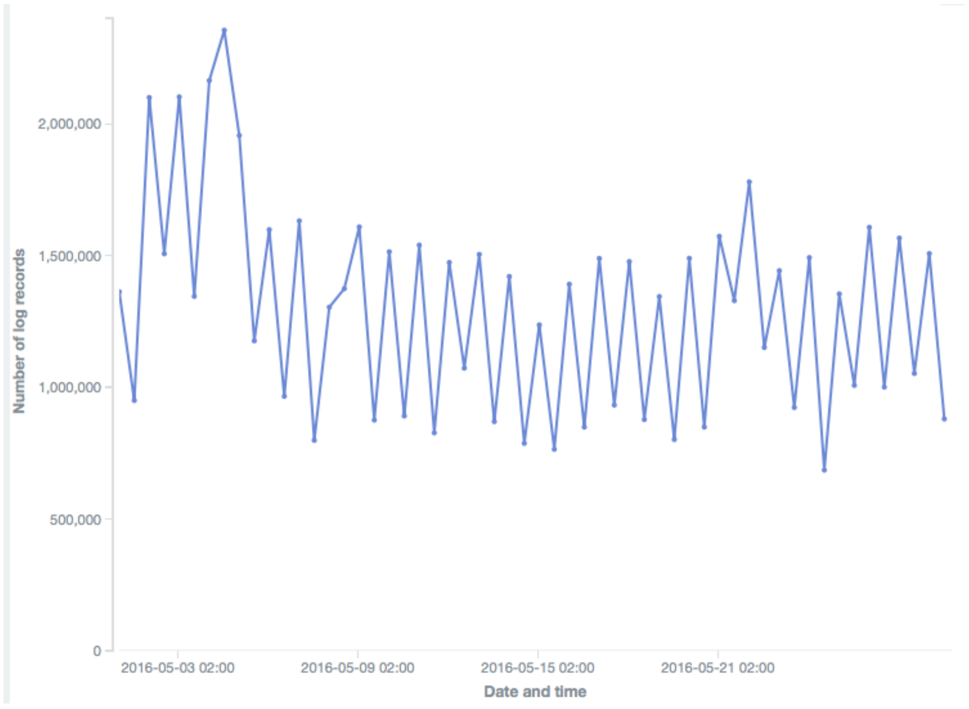

FIGURE B.2: Number of log records for application 2 aggregated per 12 hours from May 1st until May 28th.

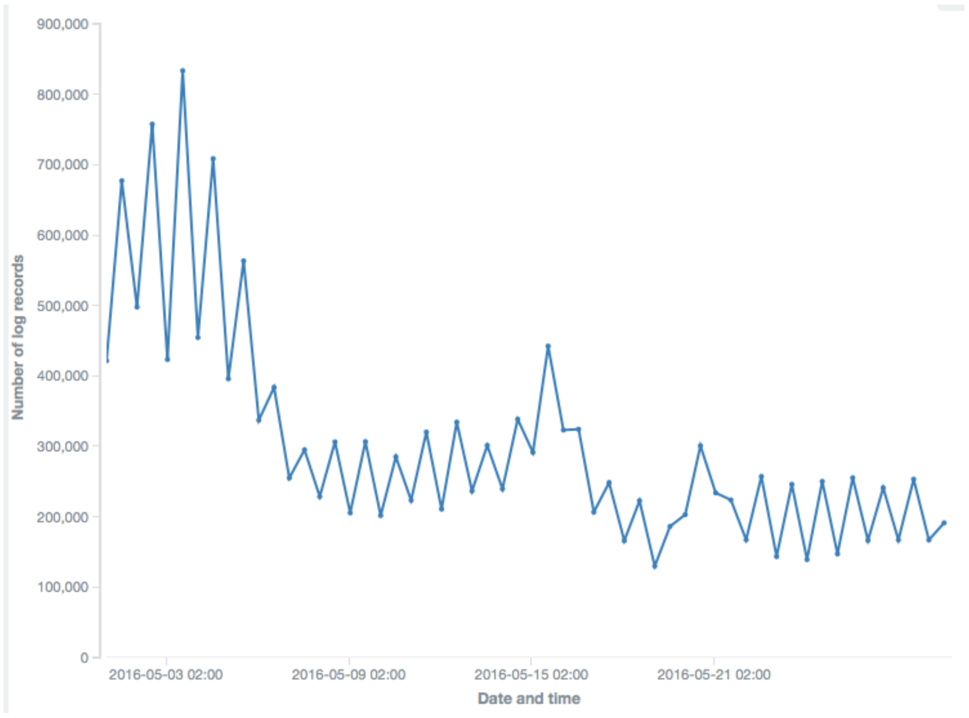

FIGURE B.3: Number of log records for application 3 aggregated per 12 hours from May 1st until May 28th.

Bibliography

[1] F. Angiulli and F. Fassetti. "Detecting distance-based outliers in streams of data". In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM. 2007, pp. 811–820.

[2] F. Angiulli and C. Pizzuti. "Fast outlier detection in high dimensional spaces". In: *PKDD*. Vol. 2. Springer. 2002, pp. 15–26.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains". In: *The annals of mathematical statistics* 41.1 (1970). JSTOR, pp. 164–171.

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "LOF: identifying density-based local outliers". In: *ACM sigmod record*. Vol. 29. ACM. 2000, pp. 93–104.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "Optics-of: Identifying local outliers". In: *Principles of data mining and knowledge discovery*. Springer, 1999, pp. 262–270.

[6] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009). ACM, p. 15.

[7] V. Chandola, A. Banerjee, and V. Kumar. "Outlier detection: A survey". In: *ACM Computing Surveys* (2007). ACM.

[8] T. Fawcett and F. Provost. "Adaptive fraud detection". In: *Data mining and knowledge discovery* 1.3 (1997). Springer, pp. 291–316.

[9] A. E. Gelfand and A. F. Smith. "Sampling-based approaches to calculating marginal densities". In: *Journal of the American statistical association* 85.410 (1990). Taylor & Francis Group, pp. 398–409.

[10] S. Geman and D. Geman. "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1984). IEEE, pp. 721–741.

[11] M. Gupta, J. Gao, C. Aggarwal, and J. Han. "Outlier detection for temporal data". In: *Synthesis Lectures on Data Mining and Knowledge Discovery* 5.1 (2014). Morgan & Claypool Publishers, pp. 1–129.

[12] V. Hautamäki, I. Kärkkäinen, and P. Fränti. "Outlier Detection Using k-Nearest Neighbour Graph." In: *ICPR (3)*. ICPR. 2004, pp. 430–433.

[13] Z. He, X. Xu, and S. Deng. "Discovering cluster-based local outliers". In: *Pattern Recognition Letters* 24.9 (2003). Elsevier, pp. 1641–1650.

[14] V. J. Hodge and J. Austin. "A survey of outlier detection methodologies". In: *Artificial Intelligence Review* 22.2 (2004). Springer, pp. 85–126.

[15] A. Ihler, J. Hutchins, and P. Smyth. "Adaptive event detection with time-varying poisson processes". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 207–216.

[16] E. M. Knorr and R. T. Ng. "Algorithms for mining distancebased outliers in large datasets". In: *Proceedings of the International Conference on Very Large Data Bases*. Citeseer. 1998, pp. 392–403.

[17] L. J. Latecki, A. Lazarevic, and D. Pokrajac. "Outlier detection with kernel density functions". In: *Machine Learning and Data Mining in Pattern Recognition*. Springer, 2007, pp. 61–75.

[18] M. Markou and S. Singh. "Novelty detection: a review—part 1: statistical approaches". In: *Signal processing* 83.12 (2003). Elsevier, pp. 2481–2497.

[19] A. Patcha and J.-M. Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends". In: *Computer networks* 51.12 (2007). Elsevier, pp. 3448–3470.

[20] D. Pokrajac, A. Lazarevic, and L. J. Latecki. "Incremental local outlier detection for data streams". In: *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. IEEE. 2007, pp. 504–515.

[21] S. Ramaswamy, R. Rastogi, and K. Shim. "Efficient algorithms for mining outliers from large data sets". In: *ACM SIGMOD Record*. Vol. 29. 2. ACM. 2000, pp. 427–438.

[22] C. J. van Rijsbergen. *Information Retrieval*. 2nd. Butterworth-Heinemann, 1979.

[23] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek. "A framework for clustering uncertain data". In: *Proceedings of the VLDB Endowment* 8.12 (2015). VLDB Endowment, pp. 1976–1979.

[24] E. Schubert, A. Zimek, and H.-P. Kriegel. "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection". In: *Data Mining and Knowledge Discovery* 28.1 (2014). Springer, pp. 190–237.

[25] S. L. Scott. *Bayesian methods and extensions for the two state Markov modulated Poisson process*. Harvard University, Deptartment of Statistics, 1998.

[26] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung. "Enhancing effectiveness of outlier detections for low density patterns". In: *Advances in Knowledge Discovery and Data Mining*. Springer, 2002, pp. 535–548.

[27] G. Widmer and M. Kubat. "Learning in the presence of concept drift and hidden contexts". In: *Machine learning* 23.1 (1996). Springer, pp. 69–101.

[28] D. Yang, E. A. Rundensteiner, and M. O. Ward. "Neighbor-based pattern detection for windows over streaming data". In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM. 2009, pp. 529–540.

[29] K. Zhang, M. Hutter, and H. Jin. "A new local distance-based outlier detection approach for scattered real-world data". In: *Advances in Knowledge Discovery and Data Mining*. Springer, 2009, pp. 813–822.