BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ANOMALY DETECTION IN SYSTEM LOG FILES USING MACHING LEARNING

DETEKCE ANOMÁLIÍ V ZÁZNAMECH SYSTÉMOVÝCH UDÁLOSTÍ

POMOCÍ STROJOVÉHO UČENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. EVA MORESOVÁ**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETR MATOUŠEK, Ph.D., M.A.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

153382

Institut:	Department of Information Systems (DIFS)
Student:	**Moresová Eva, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Software Engineering
Title:	**Anomaly Detection in System Log Files Using Machine Learning**
Category:	Networking
Academic year:	2023/24

Assignment:

1. Based on recommendation of your supervisor, research various machine learning (ML) models suitable for text anomaly detection.
2. Describe recommended datasets containing system events. Observe and analyze their contents and select features useful for behavior modelling.
3. Apply selected machine learning models on your datasets and create corresponding ML models.
4. Evaluate precision and reliability of selected ML models.
5. Compare your results with similar approaches and highlight your contribution.
6. Discuss the application of the proposed technique on large real-world log files.

Literature:

- Bhuyan, Monowar; Bhattacharyya, Dhruba K; Kalita, Jugal. Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools. 2017. ISBN:3319879685.
- Garwal, Charu C.; Reddy, Chandan K. (eds.). Data Clustering: Algorithms and Applications. CRC Press, 2014. ISBN 978-1- 46-655821-2.
- Zhao, Z., Xu, C. & Li, B. A LSTM-Based Anomaly Detection Model for Log Analysis. *J Sign Process Syst* **93**, 745–751, 2021.
- Tomaly, Radovan. *A Universal Approach for Anomaly Detection in Log Files*. Master Thesis, Charles University, Prague, 2023.
- Lv, Dan, Nurbol Luktarhan, and Yiyong Chen, 2021. "ConAnomaly: Content-Based Anomaly Detection for System Logs" *Sensors* 21, no. 18: 6125.
- Ryciak, P.;Wasielewska, K.; Janicki, A. Anomaly Detection in Log Files Using Selected Natural Language Processing Methods. Sci. 2022, 12, 5089.
- Vannel Zeufack, Donghyun Kim, Daehee Seo, Ahyoung Lee, An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis, High-Confidence Computing, Volume 1, Issue 2, 2021, 100030, ISSN 2667-2952.

Requirements for the semestral defence:
Points 1-3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Matoušek Petr, doc. Ing., Ph.D., M.A.**
Consultant:	Dr. Nabhan Khatib, AT&T
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	24.5.2024
Approval date:	30.10.2023

Abstract

Log anomaly detection is an important process that can help prevent or detect system failures, intrusion attempts and other malicious behavior. However, modern systems produce amounts of log data far beyond what is possible to analyze manually. That is why a variety of automated methods were developed for this purpose, ranging from rule based techniques to approaches using deep learning. The aim of this thesis is to compare several log anomaly detection methods to determine which one is the best suited for application on large real-world log files, represented by a collection of logs from production AAA (authentication, authorization, accounting) servers provided by AT&T. Apart from AT&T logs, the methods were applied to and evaluated on two other labeled datasets, one of which was enriched by synthetically generated anomalies. This thesis adopts three unsupervised anomaly detection methods: Local Outlier Factor, DBSCAN clustering and an OPTICS-based framework. The former two examine the logs on a sample-level, while the latter analyzes entire log sequences. All methods achieved results comparable to works with similar approaches.

Abstrakt

Detekcia anomálií v logoch je dôležitý proces, ktorý pomáha detekovať poruchy systému, pokusy o prienik do systému a ďalšie škodlivé správanie, prípadne týmto udalostiam umožňuje predchádzať. Moderné systémy však produkujú logy v množstvách, ktoré nie je možné analyzovať ručne. Preto sa na tento účel používa množstvo automatizovaných metód, od techník založených na pravidlách, až po prístupy používajúce hlboké učenie. Cieľom tejto diplomovej práce je porovnať niekoľko metód detekcie anomálií v logoch a určiť, ktorá z nich je najviac vhodná pre použitie na veľkých log súboroch z praxe. Reprezentantom takýchto dát je zbierka logov z produkčného AAA servera, ktoré boli poskytnuté firmou AT&T. Okrem AT&T logov boli metódy aplikované a vyhodnotené na dvoch ďalších anotovaných datasetoch, z ktorých jeden bol obohatený o synteticky generované anomálie. Táto práca využíva tri metódy detekcie anomálií: lokálny odľahlý faktor, zhlukovací algoritmus DBSCAN a OPTICS framework. Prvé dve metódy skúmajú logy na úrovni jednotlivých záznamov, zatiaľ čo posledná analyzuje celé sekvencie logov. Všetky metódy dosiahli výsledky porovnateľné s prácami, ktoré realizujú podobné prístupy.

Keywords

log anomaly detection, unsupervised learning, Local Outlier Factor, DBSCAN, OPTICS

Kľúčové slová

detekcia anomálií v logoch, učenie bez učiteľa, lokálny faktor odľahlosti, DBSCAN, OPTICS

Reference

MORESOVÁ, Eva. *Anomaly Detection in System Log Files Using Maching Learning.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Petr Matoušek, Ph.D., M.A.

Rozšírený abstrakt

Detekcia anomálií, nazývaná aj detekcia odľahlých hodnôt, je proces identifikácie údajov, ktoré sa výrazne líšia od bežných vzorcov správania. Tieto odchýlky môžu byť indikátorom výskytu chyby systému, škodlivých aktivít alebo jednoducho zaujímavé dáta, ktorých preskúmanie môže byť prínosné. Interpretácie sa môžu líšiť podľa oblasti, pre ktorú je detekcia anomálií použitá. Tieto oblasti zahŕňajú analýzu správania zákazníkov, detekciu finančných podvodov, monitorovanie zdravia a štatistiku, pričom pre každú existuje široký rozsah použití. Táto práca je však zameraná len na jednu z týchto domén a to na kybernetickú bezpečnosť. Anomálie odhalené v počítačových systémoch môžu byť kľúčovým indikátorom potenciálnych zlyhaní systému, útokov a pokusov o prienik do systému.

Využívanie logovania je štandardom v počítačových programoch všetkých oblastí, od operačných systémov až po aplikačný softvér. Logy sú záznamy činností, ktoré vykonáva systém ako aj udalostí, ktoré sa vyskytli počas prevádzky systému. V závislosti od systému, z ktorého logy pochádzajú môžu protokoly obsahovať údaje o procesoch operačného systému, HTTP požiadavkách na server, autentifikačné údaje užívateľov alebo vykonané databázové dotazy. Vďaka tomu sú logy vynikajúcim zdrojom informácií pre detekciu anomálií. Predpokladom je, že anomálne udalosti sa prejavia v logoch ako záznamy, ktoré sa nejakým spôsobom líšia od logov generovaných bežnými udalosti.

Jednou z veľkých prekážok pri hľadaní anomálií v logoch je veľké množstvo údajov, ktoré je potrebné spracovať. V súčasnosti počítačové systémy produkujú objem dát, ktoré je jednoducho nemožné analyzovať manuálne. Nehovoriac o tom, že tento prístup vyžaduje doménových expertov, ktorí sú podrobne oboznámení so štandardným správaním daného systému, a preto dokážu identifikovať odľahlé hodnoty. Okrem toho, niektoré anomálie nie sú ľahko rozpoznateľné, či už kvôli vedomému úsiliu útočníka alebo kvôli povahe danej anomálie.

V praxi teda nie je manuálna identifikácia realizovateľná a namiesto toho sa používajú rôzne automatizované metódy. Tieto metódy zahŕňajú prístupy založené na definovaní pravidiel, ktoré pochádzajú z techník dolovania údajov, algoritmy strojového učenia využívajúce zhlukovanie, metódy hlbokého učenia s neurónovými sieťami a ďalšie. Tieto metódy možno použiť na veľké množstvo údajov a zvyčajne bez hlbších znalostí o hľadaných anomáliách.

Cieľom tejto práce bolo vybrať metódu použiteľnú pre súbor logov poskytnutých firmou AT&T, ktoré obsahujú logy z produkčného autorizačného serveru. AT&T dataset nie je anotovaný, preto boli použité metódy, ktoré realizujú učenie bez učiteľa. Vybrané boli metódy: LOF, DBSCAN a framework pre detekciu anomálií, ktorý využíva OPTICS.

Aby mohla byť vyhodnotená účinnosť zvolených metód typickými metrikami, boli vybrané dva ďalšie anotované datasety. OPTICS framework bol vyhodnotený na HDFS datasete, ktorý obsahuje neštruktúrované textové správy. Druhý dataset, LANL, bol obohatený o synteticky generované anomálie a použitý na vyhodnotenie LOF a DBSCAN. Celkovo najvyššiu efektivitu dosiahol OPTICS framework, konkrétne F1 skóre 0,65. Ukázalo sa, že z metód aplikovaných na dataset LANL je DBSCAN účinnejší, s dosiahnutým F1 skóre 0,49. Výsledky sú porovnateľné s prácami, ktoré realizujú podobný prístup.

Metódy boli tiež aplikované na dataset od AT&T a úspešne identifikovali podozrivé správanie, ktoré sa odchyľuje od bežných vzorcov. Avšak na potvrdenie, že sa jednalo o skutočne anomálne záznamy sú však potrebné ďalšie informácie, ako napríklad história užívateľov. V čase vyhotovenia tejto práce tieto informácie neboli firmou AT&T poskytnuté.

Anomaly Detection in System Log Files Using Maching Learning

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of doc. Ing. Petr Matoušek, Ph.D., M.A. The supplementary information was provided by Ing. Nabhan Khatib, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Eva Moresová
May 22, 2024

Acknowledgements

I would like to thank my supervisor doc. Ing. Petr Matoušek, Ph.D., M.A. for his valuable advice and guidance during this work. I would also like to thank Ing. Nabhan Khatib, Ph.D. for his expertise and insightful suggestions. Computational resources for this work were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Contents

Chapter 1

Introduction

Anomaly detection, also called outlier detection, is the process of identifying data that differ significantly from normal patterns and behaviors. These deviations can be an indicator of erroneous occurrence, malicious operations, or simply interesting data points worth examining. Interpretations may vary among the many areas where anomaly detection is utilized, including customer behavior analysis, financial fraud detection, health monitoring, and statistics, each with a wide range of use cases. This work, however, is only focused on one of these domains, specifically cyber security. Anomalies detected in computer systems can be a crucial indicator of potential system failures, attacks, and intrusion attempts.

It is standard in computer programs of all areas, from operating systems to application software, to utilize logging. Logs are records of activities executed by the system and events that occurred while the system was operating. Depending on the origin, logs can contain data regarding operation system processes, HTTP requests made to a server, user authentication data or performed database queries. This makes logs an excellent data source for performing anomaly detection. The expectation is that anomalous events will be translated into logs which in some way deviate from those generated by normal events.

One of the great challenges of finding anomalies in logs is the large amount of data that needs to be processed. Nowadays, computer systems produce volumes of logs that are simply impossible to analyze manually. Not to mention that this approach would require domain experts who are familiar in detail with the standard behavior of a given system and can, therefore, identify the outliers. Furthermore, some of the anomalies are not easily noticeable, either as a result of a conscious effort from an attacker or the nature of the anomaly itself. Overall, identifying anomalies by hand is in no way a viable approach, and instead a variety of automated methods are used in practice. Approaches include rule-based methods originating from data mining techniques, machine learning algorithms using clustering, deep learning methods with neural networks, and more. These methods can be applied on large amounts of data, usually without having a deep knowledge of the anomalies present.

The focus of this work is log anomaly detection with various machine learning methods. Through collaboration with a Czech branch of the American telecommunications company AT&T residing in Brno, these methods can be selected for and applied to a specific real-world problem. The goal is to develop a log anomaly detection method, which can be applied in the case of AT&T. In order for this method to be useful and practical, the emphasis is put on two areas, the method parameters and the effectiveness. If a method requires the specification of parameters, determining their optimal value should be an automated

process. Effectiveness can be characterized by the rate of false alarms and the ability to detect present anomalies.

The remainder of this work is organized as follows. Chapter 2 introduces general concepts related to log anomaly detection, provides an overview of machine learning approaches, and a detailed description of methods used within this work. Chapter 3 describes the datasets used for evaluation of the implemented methods and their preprocessing. Chapter 4 presents the selected solutions and their implementation details. Chapter 5 contains evaluation of the achieved results. Finally, Chapter 6 examines the application of the selected machine learning methods on real-world large log files.

Chapter 2

Machine learning for log anomaly detection

Anomaly detection, also called outlier detection, is the process of identifying rare events or observations different from standard behavior. This chapter introduces the problem of log anomaly detection by defining the terms connected to it used within this work, outlining components of a general detection framework, and discussing the challenges faced in the field. This work focuses on solving anomaly detection with machine learning algorithms. Their primary classification is outlined below, including a detailed description of the methods that were utilized.

2.1 Log anomaly detection

Anomaly detection can be utilized in numerous areas, including finance and healthcare. Within the IT realm, it can be used as a part of system monitoring either to detect faults in the system or in the context of cyber-security. Logs are a natural source of information about the system runtime and its operations, so they can be analyzed for anomalies.

2.1.1 Anomaly

An anomaly, also referred to as an outlier, is a significant deviation from the normal or expected pattern of behavior [12]. Ruff et al. [43] formally define anomaly as follows: let $\mathcal{X} \subseteq \mathbb{R}^D$ be the data space of a given problem and normality be the distribution $\mathbb{P}^+$ on $\mathcal{X}$. Anomaly $x \in \mathcal{X}$ is a data point located in an area under $\mathbb{P}^+$ with low probability. In other words, if $p^+(x)$ is the probability distribution function of $\mathbb{P}^+$, set of anomalies can be defined as

$$\mathcal{A} = \{x \in \mathcal{X} \mid p^+(x) \leq \tau\}, \tau \geq 0 \tag{2.1}$$

where τ is the threshold determining the probability of $\mathcal{A}$ is sufficiently low.

Authors in [43] also highlight the difference between anomaly, novelty, and noise. All three terms signify points in low-probability areas of $\mathbb{P}^+$. However, noise is simply a rare occurrence from $\mathbb{P}^+$ while novelty is an instance from a new area of changing, evolving $\mathbb{P}^+$. In contrast, anomaly comes from a different distribution altogether. This is why anomalies are characterized as carrying significant information, unlike noise, which corrupts data and interferes with data analysis.

2.1.2 Anomaly types

Chandola et al. [12] classify anomalies into three groups: *point anomalies*, *contextual anomalies*, and *collective anomalies*. Point anomaly, also called a global anomaly, refers to a single data point that is itself anomalous. A request from a foreign IP found in network logs can be considered a point anomaly.

Contextual anomaly, sometimes called conditional, is a normal data point with respect to the dataset, but anomalous within a defined context. To detect this type of anomaly, data instances need to contain contextual as well as behavioral attributes. Contextual attribute refers to circumstances in which a log entry occurred (IP address, user, timestamp) and behavioral attribute describes the patterns exhibited within the log entry (response status, request type, number of packets). The anomalousness of two instances that are the same in terms of behavioral attributes can differ depending on their contextual features. An example of a contextual anomaly is a network traffic peak during early morning or late night hours. Although such traffic is considered normal during the day, it is anomalous in the given context of the time of the day.

When a group of related data points, which are individually normal, is anomalous because they occurred together, they are called collective anomalies. Numerous entries of failed login events can suggest occurrence of a collective anomaly. A failed login on its own is not an anomalous event; however, a large number of these events occurring together can be a mark of a brute force attack on user passwords.

2.1.3 Logs

Logs are chronological records of events that occur in a system. Log files are a collection of log entries or log lines, where each entry documents a single event. Log entries always contain a timestamp but the rest of the entry structure and content vary greatly between different systems. Log entry attributes can be one of the following types:

- categorical nominal – e.g. usernames, device identification, protocol;
- categorical ordinal – e.g. severity;
- numeric – e.g. time and date, packet size;
- unstructured text – e.g. URL, packet or website content, command, description.

2.1.4 Log-based anomaly detection framework

Generally, log-based anomaly detection frameworks include four steps: log collection, log parsing, feature extraction, and anomaly detection [23].

1. Log collection: Depending on the approach, logs from one or multiple sources are used. The logs in Figure 2.1 are collected from Hadoop Distributed File System (HDFS) running on Amazon cloud [52].
2. Log parsing: If logs from multiple sources are analyzed, there is usually a large variation in how they look. Even in logs from the same source, each type of event can contain different fields and information. The point of log parsing is to unify the log structure. Log parsing can also refer to a specific process of extracting message templates and variables from textual logs with a parser. For more information about

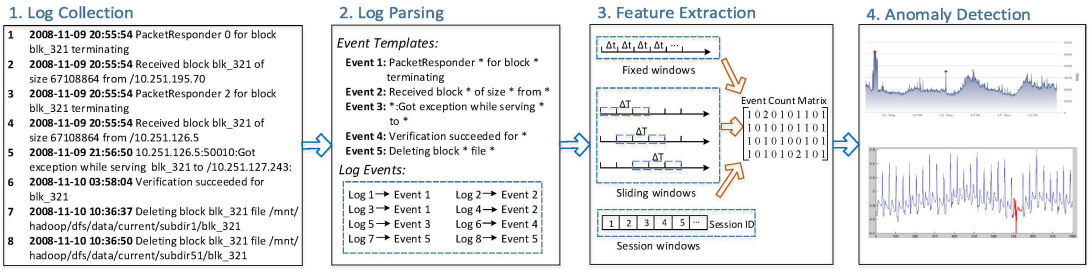

Figure 2.1: Example of general framework for log anomaly detection. Adopted from [23].

this type of parsing see Section 2.2.1 that describes the popular Drain [22] parser. Step 2 in Figure 2.1 shows the creation of templates from the collected logs.

3. Feature extraction: the feature extraction aims to transform raw data into a form suitable for a model or algorithm that is applied. This includes selection of used features (username, IP address, date), construction of new features from log templates (event count vector [54], state ratio vector [52]), or, in case of methods based on natural language processing (NLP), vectorization of log messages. Features can also be obtained by clustering. If there is a need to reduce dimensionality of a dataset, apart from removing attributes, the principal component analysis (PCA) algorithm can be used. Step 3 in Figure 2.1 shows creation of an event count vector using sliding windows.

4. Anomaly detection: the last step is applying the anomaly detection. Methods which can be applied include simple user-defined rules, more sophisticated data mining techniques, or machine learning algorithms and models.

2.1.5 Labels vs. anomaly score

There are two ways in which a method can report the result of a detection; by assigning a *score* or a *label* to a log event. Score represents the degree to which a data instance is considered anomalous, while the label only asserts whether the instance is normal or anomalous. The score can be used to determine a number of the top n anomalies with the highest anomaly score, or a threshold can be established to select the instances which should be examined further. This allows adjustments based on the characteristics of a specific problem, such as data contamination, which refers to the proportion of anomalies in a dataset.

2.1.6 Challenges of anomaly detection

Anomaly detection is based on discerning behavior in the training data in order to learn characteristics of the normal and anomalous. There are several factors that stem from the nature of anomalies that make this process a challenge. Pang et al. [39] and Ruff et al. [43] discuss the following:

- Unknowness: anomalies represent unexpected behavior, such as errors or attacks. They remain unknown until they appear.

- Diversity of anomalies: as a result of the irregular nature of anomalies, there may be a large difference in their manifestation. Different errors or attacks do not produce similar logs. This also connects to anomaly types. A collective anomaly which is a group of individually normal log entries has very different characteristics from a point anomaly which is a single anomalous entry. Therefore, it is complicated to build a model that will be able to detect all anomaly types. There are methods that try to sidestep the diversity problem by only modeling the normal class and marking any unknown behavior anomalous. This is not a perfect solution either, as discussed in the next point and in Section 2.2.3.

- Variability of normal data: normal behavior of systems is no less complicated than the anomalies that can occur. The more complex a system is, the higher the chance is that a rare normal event could be wrongly marked as an anomaly.

- Class imbalance: anomalies occur rarely, creating highly imbalanced datasets. As discussed in more detail in Section 2.2.2, some methods do not handle uneven class distributions well.

Other challenges include:

- Score threshold. For methods with anomaly score output, it is necessary to explicitly set a threshold (τ in Definition 2.1) that distinguishes anomalies from normal data. Attributes that would influence the value, such as contamination of the data, are usually unknown, so it is not a trivial task to decide what is a "sufficiently" large deviation from normal.

- Variable log structure. Even though logs are semi-structured data, there is no one standard log message format. From system to system, logs vary in content and form. Logs from a Java application might have a very different logging style to an Nginx server logs. This renders universal solutions less effective and requires tailoring to a specific problem, which is costly.

2.2 Machine learning methods

Machine learning uses models and algorithms to solve problems without being explicitly programmed to do so [45]. Solutions are predicted according to information extracted from the provided data. Machine learning methods for log anomaly detection learn to identify anomalous (normal) data instances among the log data, without explicitly defining what anomalies (normal data) look like.

Application of machine learning to a problem entails several steps. First, it is necessary to prepare the data to a form suitable for the chosen algorithm or model. Then the method itself can be applied. In the end, the results need to be evaluated. There is a plethora of methods to choose from for each of these steps. Following sections detail the techniques used within this work. Machine learning anomaly detection methods can be divided into three main categories: *supervised*, *semi-supervised*, and *unsupervised*. Figure 2.2 displays an overview of the classification, the methods that were used are highlighted in bold. The dataset provided by AT&T for this diploma project is unlabeled, so this work focuses mainly on unsupervised methods.

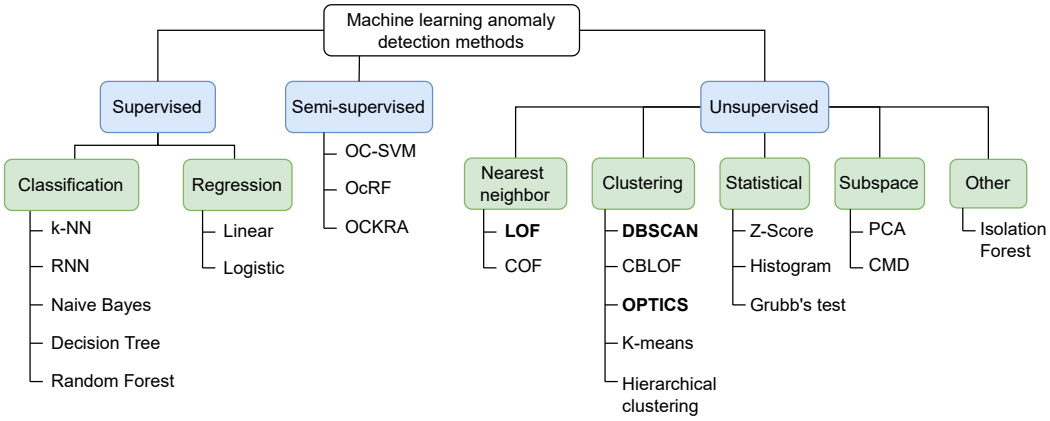

Figure 2.2: Classification of machine learning methods for anomaly detection. Methods used in this work are in bold.

2.2.1 Data preparation

Data preparation is an important step as it has a significant effect on the models ability to learn. It includes components such as:

- Data integration: To detect certain anomalies, it may be necessary to analyze logs from multiple sources. These logs need to be integrated into a dataset with unified structure.
- Data cleaning: Includes handling the missing values, noise and any redundancy created by data integration. In the case of anomaly detection, data cleaning has to be considered very carefully, so that meaningful anomalies are not removed in the process of cleaning noise.
- Data reduction: Data can be reduced in terms of numerosity (number of samples), dimensions or cardinality of attributes (number of unique values of an attribute). To decrease numerosity, only a certain time window of the original log file can be used or the logs can be sampled. To reduce dataset dimensions, the most representative features are carefully selected. New features can be extracted either manually or with more sophisticated methods such as log parsing or autoencoder models. To remove dimensions with insignificant variability there are methods such as principal component analysis (PCA) for numerical attributes or multiple correspondence analysis (MCA) for categorical variables. To decrease the cardinality of an attribute, the values can be discretized.
- Data transformation: Its role is to change data into a form digestible for the application of a machine learning algorithm. For categorical features, transformation involves encoding the nominal or ordinal values to numerical. Numerical attributes are typically transformed by normalization. Popular transformation techniques include one-hot encoding, integer encoding and Min-Max normalization.

Encoding of categorical variables

There are several encoding methods, including *one-hot encoding* and *integer encoding*. One-hot encoding, also called 1-of-n encoding, creates a new binary attribute for each category of the transformed variable. The attribute which represents the original value of a sample then has a value of one, and the rest of the attributes are zero. Integer encoding simply replaces each unique category of the variable with a number. It is good practice to assign the specific number in a way that avoids bias based on the order of appearance. Each approach has different advantages and drawbacks. Integer encoding method is simple and does not increase the dimensions of the dataset. However, when applied to nominal data, it could artificially create an ordinal relationship between the values leading to a decrease in the performance of the applied model [10]. In that way, one-hot encoding is much better at representing the distinctiveness of categories but it drastically increases the dataset dimensions if applied to variables with a large amount of unique values.

Multiple correspondence analysis

MCA is a dimensionality reduction method designed for categorical attributes based on correspondence analysis (CA) [1]. The attributes are transformed into an indicator matrix, which essentially means applying one-hot encoding on the attributes. Standard CA is applied to the matrix, creating set of row and column factor scores, whose variance corresponds to their eigenvalues [1]. To reduce dimensions data to m dimensions, factors with the m largest eigenvalues are selected and the dataset is projected onto these factors. In practice, the number of target dimensions is typically not chosen explicitly. Rather, it is determined by a required minimum cumulative variance of used factors.

Min-Max normalization

Some algorithms, such as k-means clustering, perform better on normalized data [40]. Otherwise, variables with higher magnitudes may overshadow variables with lower values [15]. That can be avoided by transforming values to the same interval. It should be noted that for certain distance measures, such as Euclidean, normalizing a dataset is necessary [2]. One of the fundamental methods, Min-Max normalization, linearly transforms data. Let A be an attribute with a value range of $[min_A, max_A]$. Min-Max normalization transforms $a_i \in A$ to a'_i in range $[new\_min_A, new\_max_A]$ [21] by:

$$a'_i = \frac{a_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A \tag{2.2}$$

In case when attributes are transformed to a new range of [0,1], the mapping function can be simplified to:

$$a'_i = \frac{a_i - min_A}{max_A - min_A} \tag{2.3}$$

Drain parser

Drain [22] is a streaming log parser with a fixed depth tree. In general, a parser processes raw log lines into structured log messages. The parser identifies logs that describe the same operation and derives a message template, i.e. the constant part, for each group of such log

messages. The variable parts, message parameters, which change from message to message, contain specific information about system runtime [22].

Drain processes raw logs in a streaming manner, one by one. When a log entry is parsed, it is matched to a most suitable group of already parsed logs or a new group is created. Drain searches for log groups using a fixed depth parse tree. Internal nodes of this tree contain rules for finding a suitable log group and leaf nodes contain a list of log groups. Before parsing, Drain preprocesses logs with simple regular expressions created by the user based on domain knowledge, called masking patterns. These regular expressions represent frequently used variables, such as IP address, and if matched, the variable is removed. This helps the parser to identify variable parts of a log message more effectively.

2.2.2 Supervised anomaly detection

Supervised anomaly detection models require the existence of training data with annotations, which means that data instances are labeled as normal or anomalous. Building a model consists of creating a function to map input to output values [34]. This function is improved with learning through feedback from a loss function. Supervised methods can perform two tasks, *regression* or *classification*. Classification models map the input values to a set of discrete output values (classes) [37]. Regression models predict the output value of a continuous dependent variable based on a set of independent variables [17].

Both tasks can be used for log anomaly detection. Application of classifiers is very straightforward; the model is trained to classify the input data, in this case the processed log files, into normal or anomalous classes. Commonly used supervised classifiers are k-nearest neighbors (KNN), support vector machines (SVM), decision tree, random forest, and supervised neural networks. When using regression models, the input independent variables become features extracted from the logs. The target variable can be the anomaly label [35] or another variable present in the source data. In the latter case, the anomalousness is based on a comparison of the predicted value with the actual value [19].

Although the supervised approach can often produce more accurate results, these methods also have several drawbacks. The most obvious one is the need for labeled data. This limit is highlighted in an area such as log anomaly detection, since labeled logs are not commonly available. The resulting model is also unable to detect novel or unseen abnormalities. This may reduce the effectiveness of the model, as new malicious behavior is always arising and evolving. Training data may lack certain groups of anomalies if they are difficult to obtain, such as in the case of very obscure attacks and system errors. Another well-known classifier issue is the processing of imbalanced class distributions [26, 32, 5]. In anomaly detection this issue is always present as there is naturally a higher amount of normal instances present in the dataset compared to anomalous objects.

2.2.3 Semi-supervised anomaly detection

Semi-supervised anomaly detection methods assume the existence of data labels for one class only. It is difficult to collect samples of anomalous behavior, so most labeled data comes from the normal class. During training, a model is built to recognize this class. It is then used to determine whether a new data point is an instance of the recognized class, and therefore normal, or deviating from the model, marking it an anomaly. This approach has the benefit of decreasing the cost and effort of labeling. It is also useful in instances where obtaining anomalies is challenging. Potential limitations include the dependence on the representativeness of the training data.

2.2.4 Unsupervised anomaly detection

Unsupervised anomaly detection does not require any labeled data. Instead, it relies on the assumption that anomalous data points differ from normal instances and are far less present in the data. This makes the methods widely applicable and allows for detection of evolving anomalies. Potential challenges of this approach are the difficulty in evaluating the accuracy of models and the interpretation of the results. Another different approach to unsupervised anomaly detection is using semi-supervised models on unlabeled data. In that case, it is crucial that the data contamination is low and that the model used is robust enough to handle the present anomalies.

There are many types of unsupervised methods, Nguyen et al., 2016 [38] mention the following categories: *Nearest-neighbor based techniques*, *Clustering based techniques*, *Statistical techniques* and Goldstein et al., 2016 [20] add an additional group of *Subspace techniques*.

Nearest-neighbor techniques

Nearest-neighbor techniques are based on the notion of density of a neighborhood of data points. The underlying assumption in these methods is that the data in more densely populated regions is normal and that the sparse data is abnormal. To determine the density of a neighborhood, a distance measure is used, which can be interpreted as the similarity between data points. There are two ways in which the assignment of an anomaly score to data points can be approached. A point can be represented by:

- the distance to its k-th nearest neighbor;
- the relative density of its neighborhood calculated by taking into account the anomaly score of each of the k nearest neighbors.

the resulting value is then compared to a predetermined threshold, and the data instance is characterized as normal or anomalous. The advantage of this group of techniques is that no assumptions about the generative distribution of the data is made. Additionally, adapting these methods to different data types is relatively straightforward. The downsides include the computational complexity of $\mathcal{O}(n^2)$. If the data is complex, it can also be rather difficult to define an appropriate distance measure.

Local Outlier Factor

Local Outlier Factor (LOF) algorithm was introduced by Breunig et al., [9]. LOF assigns a degree of outlierness to each object based on the ratio of density of the object to the average density of the neighborhood of the object. This concept is loosely related to density-based clustering such as DBSCAN [18] or OPTICS [6], however, does not involve any notion of clusters.

Formally, let *k-distance* of point A be the distance from A to its k-th nearest neighbor. *K-neighborhood* of A, denoted $N_k(A)$, is a set of points within a circle, with radius of *k-distance*(A) or $N_k(A) = \{B \mid d(A, B) \leq \textit{k-distance}(A)\}$. This also implies that the number of points in $N_k(A)$ may be greater than k. *Reachability distance* from point A to point B is defined as

$$rd_k(A, B) = max\{\textit{k-distance}(B), d(A, B)\} \tag{2.4}$$

That is, if point A is further than the k-distance of B, reachability distance is the distance between these points; otherwise, all points within the k-neighborhood of B have the reach-

ability distance value of k-distance of B. This is done to lower the statistical fluctuation of $d(A, B)$, when B is close. *Local reachability density* of point A is defined as

$$lrd_k(A) = 1/\left(\frac{\sum_{B \in N_k(A)} rd_k(A, B)}{|N_k(A)|}\right) \tag{2.5}$$

In other words, the inversion of the average reachability distance of the k-nearest points of A. *Local outlier factor* of point A is the average of the ratios of local reachability density of k-nearest points of A and point A's own local reachability density.

$$LOF_k(A) = \frac{\sum_{B \in N_k(A)} \frac{lrd_k(B)}{lrd_k(A)}}{|N_k(A)|} \tag{2.6}$$

An object with $LOF_k < 1$ has higher density than its neighbors, implying that it is an inlier. If the object has $LOF_k \sim 1$, density of its neighbors is similar to its own. An object with $LOF_k > 1$ has lower density than its neighbors and therefore can be considered an outlier.

From the above definitions, it can be seen that the value of LOF is based on one parameter k, denoted *MinPts* in [9]. *MinPts* is the number of nearest neighbors that define the local neighborhood. The LOF value does not change monotonically with changing *MinPts*, so the authors propose using a range of values. Of the resulting LOF scores, the maximum should be considered to emphasize the case where the object is the most outlying. The lower bound of the range can be interpreted as the minimum number of objects in a cluster C in order to classify an object as an outlier with respect to C. It is recommended that the lower bound be at least 10 to avoid statistical fluctuation. The specific value is application-dependent, but 10 to 20 is said to work well in most cases. The upper bound of the range represents the maximum number of objects that can be in a cluster, C while all objects from C are outliers.

Clustering based techniques

Clustering is a process of grouping objects into sets, called clusters, based on their similarity. The resulting clusters contain objects that are more similar to each other than to objects in other clusters [25]. Using clustering for anomaly detection, the results can be interpreted in one of the following ways. Firstly, it can be assumed that normal data belongs to a cluster while anomalies do not. Although not all clustering algorithms allow data instances not to be assigned to any cluster, methods such as density based spatial clustering of applications with noise (DBSCAN) can be used in this case. Another approach is based on the premise that normal data will form larger clusters, while the clusters of anomalies will be much smaller in size. Algorithms such as cluster-based LOF (CBLOF) are suited for this approach. Lastly, we can consider the data points located closer to the centroid of their cluster as normal and instances further away as anomalies. In this case, algorithms such as k-means or expectation-maximization (EM) are applicable.

The complexity of clustering depends on the chosen algorithm but ranges between $\mathcal{O}(n)$ (k-means) and $\mathcal{O}(n^2)$. As clustering algorithms are not primarily intended for anomaly detection, they are also not optimized for this purpose. Another problem with many of the algorithms is that they force every instance to be assigned to a cluster. The detection efficiency can also drop when anomalies that occur form clusters themselves.

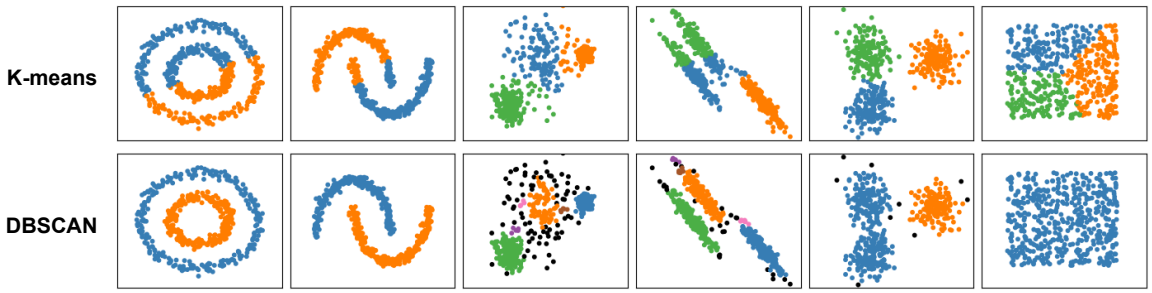

Figure 2.3: Comparison of DBSCAN and K-means clustering on different cluster shapes.

DBSCAN

Density based spatial clustering of applications with noise (DBSCAN) was proposed by Ester et al. [18]. DBSCAN's notion of clusters is based on density, meaning that clusters can be identified because the typical density of points within a cluster is higher than outside of clusters. Similarly, points can be labeled as noise if their density is lower than the density of any identified cluster. Defining the clusters in this way allows DBSCAN to detect arbitrarily shaped clusters. This ability is demonstrated in Figure 2.3 which shows the comparison between the DBSCAN and k-means, which unlike DBSCAN, forms clusters based on a distance to a centroid point.

The idea behind DBSCAN is that each cluster point should contain a specified number of points in its neighborhood of a given radius. Clusters and noise for dataset $X = \{x_1, x_2, ..., x_n\}$ of n points in a D-dimensional Euclidean space $\mathbb{R}^D$ can be formally defined as follows. Let *Eps*-neighborhood of a point $p \in X$, denoted by $N_{Eps}(p)$, be defined as $N_{Eps}(p) = \{q \in X \mid dist(p,q) \leq Eps\}$, where $dist(p,q)$ is the distance of p and q determined by the chosen distance measure. *Eps* is the user-defined radius of an *Eps*-neighborhood.

Ester et al. identify two types of points in a cluster: *core points* and *border points*. Border points have significantly less points in their *Eps*-neighborhoods. If the number of required points in a neighborhood (*MinPts*) is the same for both types of points, it would have to be set appropriately low to include the cluster borders. However, the authors point out that this would not be representative for the cluster, especially if noise is present. Instead, they define relation called *directly density-reachable*. Point p is directly density-reachable from point q when $p \in N_{Eps}(q)$ and $|N_{Eps}(q)| \geq MinPts$, for a given *Eps* and *MinPts*. Essentially, every point in a cluster has to be a member of *Eps*-neighborhood of a point which satisfies the requirements for being included in the cluster, e.g. has at least *MinPts* points in its *Eps*-neighborhood. Direct density-reachable is symmetric for two core points, but not symmetric for a core point and a border point. As an extension, let point p be *density-reachable* from point q if exists $p_1, ..., p_n, p_1 = q, p_n = p$ where p_{i+1} is directly density-reachable from p_i. Since it does not always apply that two border points are density-reachable from each other, the relation *density-connected* was defined, which says the there must be at least one core point from which the points are density-reachable. Formally, let points p and q be density-connected, if exists point o from which p and q are both density-reachable.

Based on these definitions, Ester et al. define cluster and noise as follows. *Cluster* C is a subset of points $C \subseteq X, C \neq \varnothing$, which:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p then $q \in C$.

2. $\forall p, q \in C$: p is density-connected to q.

Let $C_1, ..., C_k$ be clusters of X, with Eps_i and $MinPts_i = 1, ..., k$. *Noise* is a set of points which are not a member of any cluster C_i, $noise = \{p \in X; \forall i : p \notin C_i\}$. In an ideal case, it would be possible to define the optimal *Eps* and *MinPts* values for each cluster. Regardless, this information is not commonly available and there is no simple way to obtain it. Instead, the authors argue that it is sufficient to determine the value of the two parameters for the least dense cluster and use them for all clusters.

To construct clusters that fulfill the defined characteristics, DBSCAN iterates over all data points and processes each previously unassigned point x_i with the following:

1. Retrieve the neighbors of x_i into *seeds* (neighbors are points from $N_{Eps}(x_i)$).
2. If *seeds* contain less points than $MinPts$ (x_i is not a core point), assign x_i as `NOISE` and finish its processing. Otherwise, create a new cluster `C` and assign x_i and all *seeds* to `C`.
3. Select point s_i from *seeds* if not empty, otherwise finish the processing.
4. Retrieve neighbors of s_i into *neighbors*. If number of *neighbors* is less than $MinPts$ (s_i is not a core point), return to Step 3, otherwise continue.
5. Insert all unassigned *neighbors* into *seeds*.
6. Assign cluster `C` to all points in *neighbors* which are unassigned or `NOISE`.
7. Continue to Step 4.

Essentially, if x_i is a core point, new cluster is formed by core points density-reachable from x_i and border points density-reachable from any of the included core points.

The authors of the algorithm also introduce a heuristic for choosing the values of *MinPts* and *Eps*. For a given k, let *k-dist* be a function mapping points to the distance to their k-th nearest neighbor. *Sorted k-dist graph* is a graph of points sorted in descending order of their *k-dist*. If a point p is chosen, *Eps* is set to $k - dist(p)$ and $MinPts$ is set to k, points with equal and smaller *k-dist* become core points. Therefore, the goal is to find a point that corresponds to the maximal value of *k-dist* for the least dense cluster. The authors propose an interactive approach of choosing the value of the threshold point, as it is easier to determine visually from the sorted k-dist graph than automatically. The threshold point is located in the first so-called "valley" of the graph. Figure 2.4 shows an example of sorted k-dist graph with a threshold point p in green. All points with *k-dist* above the threshold (to the left) would be considered noise.

OPTICS

Ordering points to identify the clustering structure (OPTICS) is an extension of DBSCAN proposed by Ankerst et al. [6]. Authors of OPTICS identified several weaknesses of the existing clustering algorithms, including DBSCAN, such as the need to determine global parameters that describe the character of clusters and that the results of the clustering are highly influenced by such parameters. Both of these factors are unfavorable because it is complicated to estimate the parameters and also because the cluster density cannot be characterized globally for real-world data.

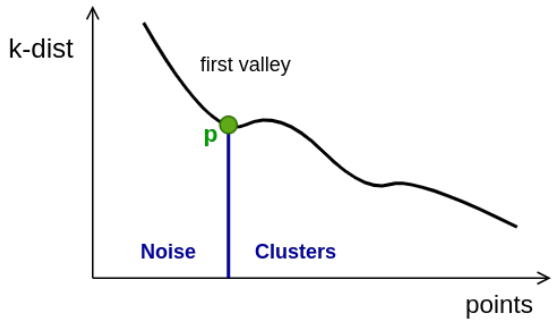

Figure 2.4: Example of a sorted k-dist graph with a threshold point p located in the first valley of the graph.

Ankerst et al. based OPTICS on the observation that for a fixed *MinPts* value, two clusters formed with a certain *Eps* value can be also identified as one cluster if the *Eps* radius is increased. The authors point out that to discover the different clustering levels, DBSCAN could be extended so that it is calculated for a variety of *Eps* values, providing a better understanding of the structures present in the data. OPTICS functions as such an extension with an infinite number of *Eps* values Eps_i up to a specified *generating distance* Eps, $0 \leq Eps_i \leq Eps$. OPTICS does not explicitly assign cluster membership to points, but rather creates an ordering of points, based on the order of processing, from which clusters can be extracted for any $Eps_i \leq Eps$. The clusters can be obtained using *core-distance* and *reachability-distance* which are calculated for each data point.

Let p be a data point from dataset D and Eps be the radius of Eps-neighborhood N_{Eps} of point p (as defined in 2.2.4). $MinPts$-dist(p) is the distance to $MinPts$' neighbor of p, $MinPts \in \mathbb{N}$. *Core-distance* of p is defined as:

$$core\text{-}dist_{Eps,MinPts}(p) = \begin{cases} \text{UNDEFINED} & \text{if } |N_{Eps}(p)| \leq MinPts, \\ MinPts\text{-}dist(p) & \text{otherwise.} \end{cases} \tag{2.7}$$

In other words, core-distance for a given $MinPts$ is the smallest distance Eps' from object o to q, $q \in N_{Eps}(p)$, for which p can be considered a core point with respect to Eps'; $Eps' \leq Eps$. *Reachability-distance* of point p to a core point o is the smallest distance, so that p is directly density-reachable from o. Formally, for a given $MinPts \in \mathbb{N}$, and Eps-neighborhood of o, $N_{Eps}(o)$, the *reachability-distance* of p to o is defined as:

$$reach\text{-}dist_{Eps,MinPts}(p,o) = \begin{cases} \text{UNDEFINED} & \text{if } |N_{Eps}(o)| \leq MinPts, \\ max(core\text{-}dist(o), dist(o,p)) & \text{otherwise.} \end{cases} \tag{2.8}$$

To illustrate these concepts, Figure 2.5 shows an example of core-distance and reachability-distance within a Eps-neighborhood(o) for $MinPts = 4$.

The OPTICS algorithm loops through points in a dataset and processes each previously unseen point p_i as follows, with respect to a given generating distance Eps:

1. Retrieve neighbors of p_i into *seeds* (neighbors are points from $N_{Eps}(p_i)$).
2. Mark p_i as processed, calculate its core-distance and output the data point and its metadata to `FILE`.

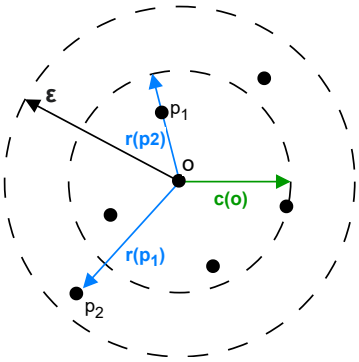

Figure 2.5: Core-distance c(o), reachability-distances r(p_1), r(p_2) for $MinPts = 4$.

3. If the calculated core-distance of p_i is undefined (p_i is not a core point), finish the processing. Otherwise, insert *seeds* into processing queue *queue*.

4. If not empty, select next point q from *queue* (point at the head of the processing queue), otherwise finish the processing.

5. Mark q as processed, calculate its core-distance and output the data point and its metadata to `FILE`.

6. If the calculated core-distance of q is undefined (q is not a core point), return to Step 4. Otherwise, retrieve neighbors of q, insert them into *seeds*, and return to Step 4.

The ordering of objects in the processing queue determines the resulting OPTICS ordering. Points in the queue are ordered by reachability-distance. If the inserted object is not in the queue yet, the point is inserted into its place based on its reachability-distance to a core point from which it was reached. If the point is already in the queue and the new reachability-distance is smaller than the previous value, the distance is updated which moves the object up in the queue. This ensures that the points are always processed with respect to the highest density (smallest reachability-distance). Result of OPTICS is a file with ordered database objects, each assigned a core-distance and reachability-distance.

From the OPTICS output is it simple to obtain a DBSCAN-like cluster assignments, with respect to a given Eps'). This extraction is performed by iterating over points in the OPTICS ordering. Each point can be identified as the first point of a cluster, as member of an already existing cluster or as noise. New cluster is created when the reachability-distance of a point is larger than the neighborhood radius Eps', meaning the point o_i is not density-reachable from the previous point o_{i-1} in the ordering, but it is a core point with respect to Eps'. If reachability-distance of a point o_i is smaller than Eps' it means it is a part of the same cluster as o_{i-1} and it is added as a member of that cluster. If reachability-distance is larger than Eps' but the point is not a core point with respect to Eps', it is simply noise.

Statistical techniques

Statistical methods use the assumption that normal data is generated from a distribution. Therefore, if a stochastic model can be fitted to the data, a probability that a data point was generated by this model can be determined for each new instance. Then, given a probability threshold, the instance is normal or anomalous. There are two types of statistical methods:

parametric, which assume the generating distribution, and non-parametric, which do not and can therefore be applied to any kind of data. The computing complexity of statistical methods is usually $\mathcal{O}(n)$.

The drawback of this approach is that an incorrect assumption of the generating distribution can degrade the results. Histogram-based methods do not take into account attribute interactions and only determine anomalies based on isolated attribute values, not rare attribute combinations.

Subspace techniques

Subspace, also called spectral, methods assume that the dimensions of data can be reduced into a lower-dimensional space in a way that will show a significant difference between normal and anomalous data instances. One of the popular methods is PCA. The advantage of this approach is that it is suitable for high-dimensional data. These methods can also be used for data preprocessing to reduce dimensions.

2.2.5 Evaluation metrics

Anomaly detection can be interpreted as a binary classification, where the classes are normal and anomalous. The most basic metric for evaluating model results is accuracy, calculated as the number of correct predictions divided by the number of all predictions. However, this metric can be misleading in the case of highly imbalanced datasets. For example, if the dataset contains only 2% of the target class and the model classifies all samples as the non-target class, its accuracy is 98% which is not representative of its performance at all. Instead, a different set of metrics must be used. When the predicted label is compared to the ground truth label, the result is one of 4 combinations depicted in a so-called confusion matrix, Figure 2.6.

		Actual	
		Positive	Negative
Predicted	Positive	True positive	False positive
	Negative	False negative	True negative

Figure 2.6: Confusion matrix.

True positive (TP) is the number of instances correctly classified as members of a class. False positive (FP) is the number of instances of incorrectly classified as members of a class. Correspondingly, true negatives (TN) are instances correctly classified as non-members of class, and false negatives (FN) are incorrectly classified as non-members. With this definition, precision, recall, and F1 score can be defined as follows:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Precision conveys how many of the predicted positives are truly positive, meaning that the higher the precision, the lower the rate of false positives. Recall states how many of

the true positives were classified as positive. Therefore, the value of 1 means that all true positives were recognized as positives by the model.

2.3 Related work

Supervised and unsupervised anomaly detection has been studied extensively. Since this work operates in an unsupervised setting, only works of that nature were examined within the scope. Clustering methods are widely utilized for unsupervised log anomaly detection. Popular algorithms include k-means, DBSCAN, OPTICS, and hierarchical clustering.

Henriques et al. [24] use k-means to identify anomalous cluster in NASA HTTP logs and then apply XGBoost decision tree for better interpretability of the results. Dani et al. [16] use k-means to cluster logs from a high performance computing facility and determine the anomalousness of a data point based on its distance from its respective centroid. The number of clusters is estimated as the number of unique event types present in the log data.

Alghamdi and Reger [3] propose an unsupervised framework for pattern extraction of multi-stage threats utilizing DBSCAN. The framework is designed to process logs from multiple sources and identify patterns within the logs in two stages. Phase one aims to identify the inner behavior of logs. Each log is processed individually and DBSCAN is used to extract clusters of samples which share the same characteristics. For this stage, the features that are unique among the sources were selected. The resulting cluster assignments are used as a new feature for the next phase. The objective of phase two is to correlate behaviors from different sources based on the their shared features, such as timestamp, user, or source address. DBSCAN is applied again, this time to construct behavior patterns, and they are assigned a signature to allow identifying similar patterns. The two step approach is interesting as it allows for a multifaceted analysis of the data from multiple sources, honoring both the inner behavior of each source and also providing insight on correlations between the different logs. However, it does not provide a clear identification of anomalous behavior and still requires a considerable amount of effort to analyze the results. It is also of note, that the used dataset was quite small, only amounting to about 80 thousand log entries.

LOF is another widely used algorithm, which is as the name suggests, performs outlier detection locally. Paulauskas and Bagdonas [41] use LOF to identify outliers in aggregated network flow logs. Cheng et al. [14] introduce a solution integrating Isolation Forest (IF) and LOF. IF is used to prune a part of the normal data points before applying LOF, to decrease the rate of false positives. They also design an algorithm to determine the threshold for pruning since IF itself does not provide such information. The proposed algorithm, however, is dependent on application specific parameters and authors provide no guidelines on how to estimate them. The previous works applied LOF in a static setting, calculating outlier scores for an unchanging dataset. There were also efforts to adapt LOF for application in a streaming environment. Alghushairy et al. [4] provide a review of the adaptations including incremental LOF (ILOF) [42], memory efficient ILOF (MiLOF) [44], and density summarizing ILOF (DILOF) [36].

Another popular notion is to base the anomaly detection in the sequential nature of logs and examine the frequency with which events occur in the data. The general outline of this kind of methods is: parse logs to extract event types (log templates), construct features that capture the frequency patterns with respect to event types, and use machine

a learning method to detect deviation from established normal patterns. The features are usually formed for log file chunks of a given size called windows.

Method proposed by Xu et al. [52] for mining console logs follows this outline. They acquire the log templates directly from the source code. During parsing, each log line is matched to a template and the message variables are extracted. Two features are created: state ratio vector and message count vector. The first captures the ratios of variable values within a window and the second contains the number of logs matching each template within a window. PCA is applied to these features to identify anomalies, and the decision tree is used to used to visualize the rules for detecting an anomaly. Though the authors used PCA, it can be replaced by any other detection algorithm making the solution more flexible. However, this approach relies on having access to all source code producing the log data which is not realistic for real-world scenarios.

LogCluster [31] includes a construction phase which uses logs from testing environment and a testing phase which uses production logs. Clustering is used to extract log templates from testing data that are transformed into weighted vectors of log sequences. These vectors are clustered with Agglomerative Hierarchical clustering and centroids of the extracted clusters serve as representative patterns. The logs from production are also converted into vectors and clustered. Any patterns that are new compared to the ones extracted from test environment are to be examined as potentially suspicious.

Zeufack et al. [54] combine approaches of [52] and [31] into a two stage framework. During the first phase, called knowledge base construction, behavior patterns are extracted. Log are parsed into templates with a fixed-tree parsing algorithm Drain and the templates are used to construct event count vectors, which capture the number of logs of a certain template present in a given windows. The vectors are clustered using OPTICS and centroids of the identified clusters serve as a knowledge base of normal behavior patterns. The second part of the framework is streaming anomaly detection. When a new log entry is added, an event count vector is constructed for a window containing the newest and the latest entries. The window is marked anomalous or normal based on its distance to the closest centroid in the knowledge base. This approach removes the dependence of log parsing on accessing source codes and the two stage structure enables to apply the framework in a streaming manner.

It can be noted that some works seem to share a common flaw. Even when developing an unsupervised log anomaly detection method, it is necessary to use annotated data to accurately evaluate the efficiency and precision of the proposed method. The number of publicly available labeled datasets is limited, so many papers test their frameworks on a select few, such as the widely popular Hadoop Distributed File System (HDFS) log data set [52]. This causes many of the methods to be applicable to a specific type of logs and it can be challenging to adapt them for other real-world data.

2.4 Summary

To summarize, anomalies are deviations from normal behavior or patterns. There are three types of anomalies: point anomalies, collective anomalies, and contextual anomalies, and each type manifests differently in log data. Anomalies can be detected using machine learning methods, and this work focuses specifically on unsupervised methods as the goal is to apply them to an unlabeled dataset. The algorithms can be grouped based on their underlying principles into nearest-neighbor based, clustering, statistical, and subspace techniques. Algorithms from the former two were selected for implementation in this work, inspired by

the popularity of the methods in other log anomaly detection works. Namely, the algorithms LOF, DBSCAN and OPTICS were utilized. Chapter 4 describes their application on the chosen datasets in detail.

Chapter 3

Description of used datasets

This chapter contains a description of the datasets used in this work and their preprocessing. The use of multiple datasets was motivated by the lack of labels in the dataset provided by AT&T. To use evaluation measures that require identified ground truths, it was instrumental to select other data sources with labels. The idea was to choose another dataset similar to AT&T, when accounting for the source of the data and the nature of the attributes, to ensure that a matching approach to data preprocessing and feature selection and extraction could be implemented for both. The AT&T dataset contains structured data of categorical and numerical nature, as well as an attribute with an unstructured text. It was challenging to find a single similar dataset so instead, two publicly available datasets were selected. First one is the the Hadoop Distributed File System dataset [52] which contains unstructured text messages and was used to evaluate the methods that expect this kind of input. The second selected dataset is the Los Alamos National Laboratory cyber security dataset [27] which only contains categorical attributes and was used to evaluate approaches suitable for structured data. Table 3.1 summarizes the basic characteristics of each dataset, including the size, number of lines (Entries), the time span and a number of attributes of a dataset if applicable.

Table 3.1: Statistics of used dataset files.

Dataset	Size [MB]	Entries	Time span	Attributes
HDFS	355.01	500,000	14:56:27	-
LANL	62.16	945,202	28 days 10:14:48	9
AT&T	103.85	867,697	14 days 03:44:18	12

3.1 Hadoop Distributed File System dataset

Apache Hadoop[1] is an open source framework designed for handling large files and distributed processing. Hadoop Distributed File System (HDFS) is the storage component of Hadoop. HDFS dataset [52] contains of logs generated by HDFS collected from more than 200 Amazon cloud nodes. The logs had been manually labeled. This work used the HDFS version maintained by Loghub [56], namely HDFS_v1. Out of the total 11,175,629 logs lines, 500 were used, which span approximately 15 hours. HDFS log entry consists of

[1]Apache Hadoop: Distributed storage and processing of large datasets. Available at: https://hadoop.apache.org/

a header and a log message. The header contains date, time, process ID, logging level, and identification of a component that generated the entry. Message contains the contents of log in a form of an unstructured text. Following is an example of raw HDFS logs:

```
081109 203519 145 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block
blk_-1608999687919862906 terminating
081109 203519 145 INFO dfs.DataNode$PacketResponder: Received block blk_-1608999
of size 91178 from /10.250.10.6
```

Each entry is tied to a specific block and all logs corresponding to one block form a trace that was labeled as anomalous or normal by domain experts. To obtain labels for individual samples, if a log entry belongs to an anomalous block trace it was labeled as an anomaly and otherwise as normal. HDFS dataset contains around 3% of anomalies.

3.1.1 Data preparation

The header of HDFS logs was discarded, only keeping the contents of the log message, similarly to other works [54, 33, 7]. The log messages were parsed with Drain parser. Drain first preprocesses the input data with regular expressions, called masking patterns. Tokens that match any masking pattern are replaced by a specified placeholder. Three masking patterns were used:

- block ID pattern – to mask block IDs, e.g. `blk_-1608999687919862906`, with `<blk>` placeholder token,
- path pattern – to mask paths, e.g. `/mnt/hadoop/mapred/system/job_80/job.jar.`, with `<path>` token,
- location pattern – to mask locations, e.g. `/10.250.19.102:54106`, with `<loc>` token.

After preprocessing, the log messages are parsed. During parsing, Drain distinguishes between the constant and variable parts of a log message. The variable part includes message parameters that capture the runtime state of a system and are different from log to log (block ID, size of block). The constant part is identical for logs that document the same system operation (creating a block) and it represents the message template. Figure 3.1 shows the processing of a raw HDFS log line into a message template and parameters.

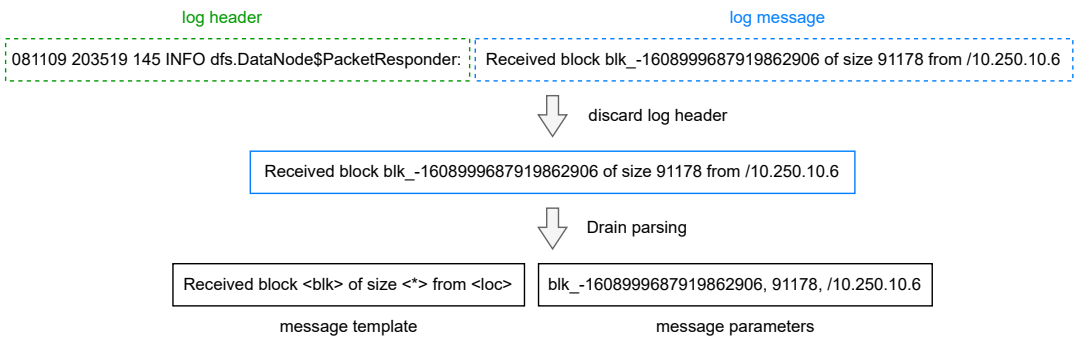

Figure 3.1: Processing and parsing of a raw HDFS log line, using Drain parser.

In total, 33 different templates were extracted from the HDFS logs. The message parameters extracted by Drain were discarded and only the log templates were used. Each

template was replaced by a numerical identifier. In conclusion, the preprocessing transformed the input raw logs into a sequence of log template IDs.

3.2 Los Alamos National Laboratory dataset

Los Alamos National Laboratory (LANL) cyber security dataset [27] contains log events from their internal network collected from multiple sources over 58 days. In total, the dataset contains more than 1.5 billion events captured in authentication records, Domain Name Service (DNS) lookups, network flow data and others, along with a set of events generated by red team attacks.

In this work, only the authentication log file was used, which contains Windows-based authentication events. The original file was more than 70 gigabytes with over a billion log lines. To avoid high computational complexity, the number of records was reduced. The red team attacks did not span over the entirety of the 58 days, so the data was cropped to the time frame of the attacks. After applying random sampling, the log file contained approximately 2.4 million events. Following Tuor et al. [50] all computer-to-computer communication was removed, keeping only the interactions involving users. The final log file contained approximately 900 thousand samples spanning 28 days, of which 749 events were anomalous.

Each line in the log file describes an authentication event with attributes such as the time of the event, users, computer identification, authentication type and orientation. All attributes are listed in Table 3.2 along with the number of their unique values (Unique), number of missing values (Missing), an example value, and the data type of the attribute.

Table 3.2: Description of LANL authentication log file attributes.

Attribute	Unique	Missing	Example value	Data type
time	760,492	0	2568342	integer
src_user	23,807	0	C104$@DOM1	enum
dst_user	24,760	0	ANONYMOUS LOGON@C586	enum
src_computer	12,670	0	C586	enum
dst_computer	11,377	0	C988	enum
auth_type	15	550,166	Kerberos	enum
logon_type	9	179,013	Network	enum
auth_orientation	7	0	LogOff	enum
success	2	0	Success	boolean

3.2.1 Data preprocessing

Attribute `src_user` had a `user@domain` format. These values were split by `@` separator into user and domain and stored in `src_user` and `src_domain`, respectively. For example, the value `U2281@DOM1` was transformed into `{ src_user:U2281, src_domain:DOM1 }`. Attribute `dst_user` was transformed in the same way into `dst_user` and `dst_domain`.

Attributes `auth_type` and `logon_type` had a significant amount of missing values. However, they capture information that could be valuable for determining anomalous events, so they were not removed. Instead, the missing values were replaced by a new category called

"Other". When inspecting the attributes further, it was discovered `auth_type` contains the following values (among others):

```
MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
MICROSOFT_AUTHENTICATION_PACKAG
MICROSOFT_AUTHENTICATION_PACKAGE_V1_
MICROSOFT_AUTHENTICATION_PACKAGE
MICROSOFT_AUTHENTICATION_PACKAGE_
MICROSOFT_AUTHENTICATION_PACK
MICROSOFT_AUTHENTICATION_PACKA
MICROSOFT_AUTHENTICATION_PACKAGE_V1
MICROSOFT_AUTHENTICATION_PACKAGE_V
MICROSOFT_AUTHENTICATION_PAC
```

Rather than being an anomaly caused by malicious behavior, the differing values seemed more like a mistake made during data collection. Therefore, all the variations were replaced by `MICROSOFT_AUTHENTICATION_PACKAGE`.

It was up for consideration whether to include the timestamp of the logs. Collective anomalies are entirely dependent on the context of their time frame and cannot be detected otherwise. On another hand, the provided logs spanned only a few weeks period, which might not have offered enough data to form reliable patterns of normal log frequency on particular days of the week or in specific hours. Instead, the inclusion of these features could have reduced the effectiveness of detection by introducing unnecessary noise. The patterns connected to time are discussed further in Section 3.2.2 and the Figure 3.3 provided there suggests, that patterns with regard to time did exist, but it was not possible to ascertain how representative the patterns were. In the end, the time data was not used.

The next step was to transform the values into numerical. The LANL dataset contained mostly categorical attributes, specifically categorical attributes with extremely high cardinality (number of categories). If they were encoded with one-hot encoding, the resulting dataset would have an unmanageable number of attributes. Applying anomaly detection on such data would not only require a substantial amount of resources (memory and computing time), but could also lower the efficiency of methods if they did not handle sparse data well. So instead, a simple integer encoding was used with a randomization of the assigned values to avoid the bias based on the order of appearance. Note that this may have negatively affected the efficiency of detection by introducing ordinal relationships between categories. Integer encoding was applied to `src_user`, `dst_user`, `src_domain`, `dst_domain`, `src_computer`, and `dst_computer`. The values in the dataset had been unified across all records, meaning user `U1` in `src_user` and `U1` in `dst_user` represent the same user. Similarly, the same value in `src_computer` and `dst_computer` symbolizes one particular machine. To preserve the ties between the values, all six transformed variables were encoded by the same mapping. First, the columns were joined and duplicates were removed. Then, the integer encoding was applied, creating a mapping of each value to a unique number. The mapping was then applied to the values in the original columns.

The remaining categorical attributes, `auth_orientation`, `auth_type`, and `logon_type`, were encoded with one-hot encoding creating 25 columns. MCA was applied to reduce the dimensions, keeping at minimum 95% of the original variance, which yielded 17 attributes.

The final preparation step was to normalize the values. All attributes were normalized with Min-Max normalization to a range $[0, 1]$ using Equation 2.3.

3.2.2 Generating anomalies

The LANL dataset is labeled and contains 749 events classified as anomalies. All anomalous events came from one of four source computers, presumably the devices the attackers used. However, without any further context, the activity was not suspicious compared to the rest of the logs. The anomalous log entries all had the same values of `auth_orientation`, `auth_type`, and `logon_type`, but these values were not unique to the anomalies and could be found in normal data as well. The number of events associated to the attacker computers also did not stand out, as the data contained many computers that appeared less often or with a similar frequency. The pairing of `src_user` and `src_computer` was N-to-N, meaning that one user was associated with multiple computers within the dataset and one computer to multiple users. So, it was not an exception that the attacker computers were connected to different users.

In conclusion, the anomalies presented in LANL dataset were not representative as a deviation from normal behavior or patterns. Therefore, to test the power of the selected anomaly detection methods in a more meaningful way, the dataset was enriched by generated anomalies. Anomalies of each type 2.1.2 were generated to allow for testing of the robustness of detection methods.

To create the contextual anomalies, it was first necessary to inspect if the data contained some patterns. Most of the data in the dataset was de-identified, including the timestamps, which as a result begins at an epoch of one with a resolution of one second. The de-identification took away the possibility to intuitively make assumptions about the distribution of events in time. For example, it is common to observe differences in the amount of records based on the day of the week or differences during the day and the night. Although this behavior might not be present at the expected times, the patterns could still be present as the time frame and time difference between events were preserved. Figure 3.2 shows the number of log entries per hour and day. Note that because of the de-identification, hour 12 does not correspond to noon and day zero does not correspond to Monday. However, the plot shows a clearly visible trend in the activity. Certain hours for a specific day contained more log entries, and there were differences between the activity levels of certain days. It could be inferred from the plot that days five and six are likely to be the weekend based on the lower number of logs. Similarly, hours ten through 24 seem to represent daytime because of the increased activity. Anyhow, it was not necessary to assign specific labels to the values of day of week and hour. What is important is that the data contained patterns, which when disturbed, would be anomalous.

Following is a list of created anomalies, with descriptions, their assigned label and a brief explanation of how they were generated:

1. Point anomalies:
    - `NontypAuth`: Atypical value of `auth_type`, `logon_type` and `auth_orientation` with respect to their global frequency in the dataset. Histogram of the different combinations of these attributes was retrieved and randomly selected combinations with zero occurrences were generated in a specified amount.
    - `NontypAuthUser`: Atypical value of attributes `auth_type`, `auth_orientation` and `logon_type` with respect to behavior of a particular user. Anomalies were created the same way as `NontypAuth` but for a specified user.
    - `UnexAuthUser`: Unexpected successful of failed authentication from a system user. For the system user `LOCAL SERVICE` few of the most common combinations

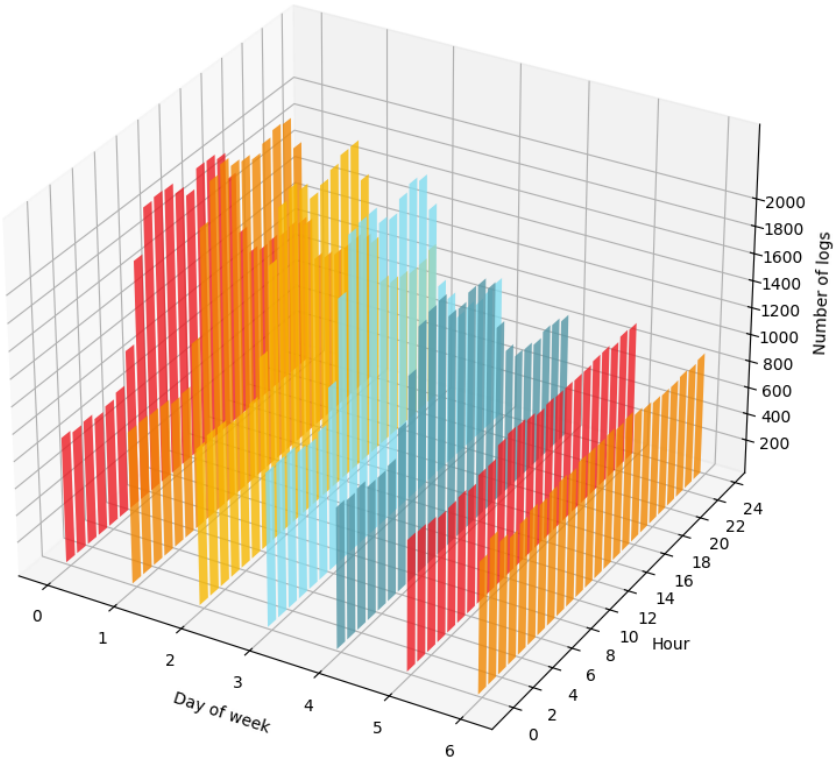

Figure 3.2: Average number of logs per hour and day in the week in the LANL dataset. The day of week and hour values are de-identified, Day of week 0 does not correspond to Monday and Hour 12 does not correspond to noon.

of attributes were chosen and assigned the opposite `Success` value. Such samples were inserted uniformly within the dataset, with respect to a specified amount.

2. Collective anomalies:

   - `ManyFailsUser`: Increase in failed authentication attempts of a particular user within a short time period. Based on a histogram of user login success, a random user from the tenth to 20th most common users was chosen. For this user were generated failed login attempts. The rest of the attributes has the most common values with respect to the selected user.
   - `ManyFailsGlobal, ManyFailsGlobalUnspec`: Overall increase in failed authentication attempts within a short time period. `ManyFail` anomalies are generated as failed login events for a random user, where other attributes have the most common values. `ManyFailUnspec` are generated as a failed login event for a random user and the other attributes have values generated with respect to their frequency in the dataset.
   - `BurstLogonUser` Multiple logon attempts of the same user within a short time period. A random user is chosen and an increased number of login events (20 to 40) is generated within windows of specified size throughout the dataset.
   - `BurstLogonAtrib`: Repeated authentication attempts with infrequent attributes. Anomalies are generated the same way as `BurstLogonUser` but with infrequent values of attributes.

3. Contextual anomalies:

   - `UnusualActivity`: Increased activity (number of authentication events) during a typically low-activity time periods (during night or the weekend). Five least frequented users were selected. Login events with random attributes were generated for the users during a low activity time period.
   - `UnusualActivityUser`: User is active in an atypical time based on their usual behavior. User, who is typically active during a low activity time period (a supposed weekend), was selected and login events were generated during his inactive days.

In total, 19,054 anomalies were generated which is approximately 2% of the whole dataset. The same number of anomalies was created for each type.

3.3 AT&T dataset

AT&T provided a dataset from one of their servers that contains more than 800 thousand log entries spanning approximately two weeks. This dataset consists of log records from an AAA (authentication, authorization, accounting) server. AAA servers are used to handle user access to networks and network resources by providing centralized management of credentials and user group administration. As such, the log files from these severs have a great potential to carry important information concerning system security, so they can be invaluable when detecting attacks. The exact properties of the log file are displayed in Table 3.1.

The log file contains various attributes which describe a logged event, most importantly the timestamp, source and destination IP addresses, username, command type, and

Table 3.3: Description of AT&T AAA server log file attributes.

Attribute	Unique	Missing	Example value	Data type
entry_ts	172,300	0	2023-10-12 06:52:07	datetime
username	50	2,322	e02916	enum
cmd_authorized	2	349,833	YES	boolean
cmd_type	5	0	TACACS+_LOGIN	enum
ip_dest	484	0	0.0.0.2	enum
ip_src	20	0	0.0.1.231	enum
customer_name	2	0	Customer 0	enum
aaa_server	1	0	0.0.1.245	enum
server_port	2	0	49	integer
cmd	2,116	0	terminal length 0	string
auth_mode	3	753,867	PASSWORD	enum
total_packet_rt	1,408	349,833	104	integer

the command itself. The list of all attributes with the number of unique values (Unique), number of missing values (Missing), an example value and a data type are shown in Table 3.3.

3.3.1 Data preprocessing

The `cmd` attribute of some samples contained multiple commands separated by semicolons. These samples were split into multiple rows, where each row contained one of the commands and the rest of the attributes were duplicated. For example:

```
1: 2023-10-10 16:47:10, NetBrain, TACACS+_ACCOUNTING, 0.0.1.221, 0.0.1.231,
   configure terminal;vrf context PreprodVRF
                                    ↓
1: 2023-10-10 16:47:10, NetBrain, TACACS+_ACCOUNTING, 0.0.1.221, 0.0.1.231,
   configure terminal
2: 2023-10-10 16:47:10, NetBrain, TACACS+_ACCOUNTING, 0.0.1.221, 0.0.1.231,
   vrf context PreprodVRF
```

`Cmd` contained Cisco Internetwork Operating System (IOS) commands used for configuring and troubleshooting network equipment. The number of unique values of `cmd` shown in Table 3.3 was very large, however, this statistic did not take into account that the same command can be used with different arguments values. For example, `terminal length 0` and `terminal length 55` are the same command for configuring the terminal, but during the analysis they were treated as two unique commands. To get a more general outlook on what commands were being used, Drain parser was used to extract command templates from `cmd`. Drain was used with masking patterns for file paths, IP addresses, and other generic patterns such as numbers. Commands were parsed into 218 templates. New attribute with the command template ID, called `template` was created and the original `cmd` attribute was removed from the dataset.

All data had been collected on the same server and the `aaa_server` IP address was the same for all entries, so it was removed. The dataset also contained attributes with a large number of missing values: `cmd_authorized`, `auth_mode`, and `total_packet_rt`. The missing values were connected to the content of `cmd_type`, mostly attributes were absent for command type `TACACS+_ACCOUNTING` and `auth_mode` values were absent for all

`TACACS+_COMMAND_AUTHOR` commands. As these attributes were not common for all entries, they were omitted. Additionally, all rows where `username` was missing, which was around two thousand, were removed. The `customer_name` corresponded to the `server_port` attribute, meaning that a customer always communicated on the same port, so the port attribute had been removed as redundant.

As for the time attribute `entry_ts`, the same argument as in Section 3.2.1 can be made, whether the used time span is sufficient to reliably characterize any patterns, since the provided logs spanned only a two-week period. An identical approach was adopted and the timestamp was not used.

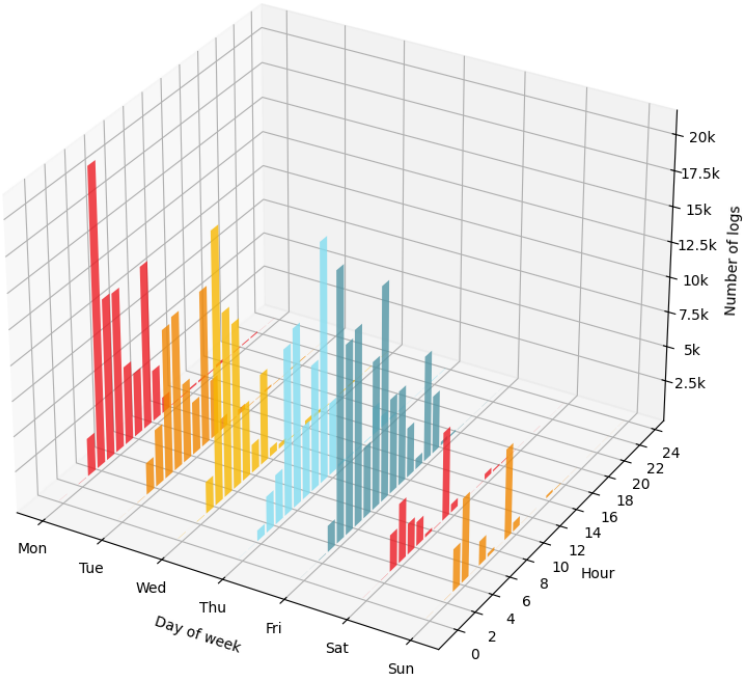

Figure 3.3: Average number of log messages per hour and day in the week.

The next step had been to convert categorical values into a numerical representation. The attributes `ip_dest`, `template`, and `username` had a large number of categories, i.e. high cardinality. If they were encoded with one-hot encoding, the resulting dataset would have more than 500 attributes. Similarly to the LANL preprocessing, integer encoding was used. The remaining categorical variables; `cmd_type`, `ip_src`, and `customer_name`, were encoded using one-hot encoding creating 27 binary attributes. MCA was applied to reduce the dimensions down to 21 attributes that preserved at least 95% of the original variance. As a final preparation step, the values were normalized with Min-Max normalization to a range $[0, 1]$ using Equation 2.3.

3.4 Summary

Three datasets were selected for application of the anomaly detection methods; AT&T server logs dataset and LANL dataset with authentication records and HDFS dataset with logs from a distributed file system. Only one feature was used to characterize HDFS logs,

a total of six features were used to characterize AT&T dataset and ten features were used for the LANL dataset:

- HDFS – `template`;
- LANL – `success`, `src_user`, `dst_user`, `src_domain`, `dst_domain`, `src_computer`, `dst_computer`, `auth_type`, `logon_type`, `auth_orientation`;
- AT&T – `username`, `cmd_type`, `ip_src`, `ip_dest`, `customer_name`, and `template` which was extracted from the original `cmd` attribute.

AT&T and LANL datasets were transformed from categorical to numerical data using integer encoding for attributes with high cardinality and one-hot encoding for the rest of the attributes with subsequent reduction of dimensions using MCA. All attributes were normalized with Min-Max scaling to a range of [0,1] before applying the anomaly detection algorithms.

Chapter 4

Implementation of selected solutions

There are countless methods that can be utilized for log anomaly detection. Based on the core principle on which a certain anomaly detection algorithm operates, or the way it is applied, some approaches are more likely to detect point anomalies, and others are more suited to detect collective and contextual anomalies. To explore both cases, LOF, DBSCAN, and OPTICS framework were selected. LOF and DBSCAN (in the way they were applied within this work) are both more prone to detecting singular log lines and small groups of similar anomalies deviating from an established normal behavior. The size of anomaly groups that the algorithms are able to detect depends on the parameter settings. On the other hand, the selected OPTICS framework analyzes sequences of logs and is sensitive to collective anomalies, since they are very different in comparison to normal sequence patterns. This chapter details application of the selected methods, along with tools and techniques used for estimating method parameters and interpreting the results on an unlabeled dataset.

4.1 Local Outlier Factor

The first method applied for anomaly detection was LOF. It is used to calculate outlier scores for each data point and points with the highest anomaly score are labeled as anomalies. The algorithm itself depends on one parameter *MinPts*, which defines the size of a neighborhood from which the outlier score is calculated. However, to obtain labels from the scores, it is necessary to determine the expected proportion of anomalous samples in the dataset (contamination). Based on contamination value α a threshold δ can be calculated as $(1 - \alpha)$ percentile. Any samples with outlier score greater than δ are anomalous and samples with a score less than δ are normal. Therefore, in practice, the contamination value essentially becomes a second parameter of LOF.

Authors of LOF recommends using a range of *MinPts* values with guidance on how to determine application specific lower and upper bounds of the parameter range. In real-world scenarios, using a range of values for every detection would be computationally expensive, so the desire is to find one suitable *MinPts* value on subset of the dataset and use it in consecutive applications. A combination of methods was used to estimate the parameters for LOF.

4.1.1 Automatic hyperparameter tuning method

Xu et al. [53] propose an automatic hyperparameter tuning method (AT) for the joint tuning of *MinPts* and contamination. The method is inspired by the mechanics of support vector machine, as it selects parameters which maximize the difference between anomalous and normal samples around the decision boundary [53].

Algorithm 1 displays pseudocode of the proposed method. Let $grid_c$ and $grid_k$ be sets of candidate values of contamination c and neighborhood size k, respectively, and $X \in \mathbb{R}^{n \times d}$ be the input data of n samples and d attributes. Anomalies are the top $\lfloor cn \rfloor$ samples with the highest LOF score. For each combination of $c \in grid_c$ and $k \in grid_k$, the mean and variance of natural logarithm of LOF scores is calculated for top $\lfloor cn \rfloor$ anomalies and top $\lfloor cn \rfloor$ normal samples. The standardized difference between anomalous and normal samples is defined as

$$T_{c,k} = \frac{M_{c,k,out} - M_{c,k,in}}{\sqrt{\frac{1}{\lfloor cn \rfloor}(V_{c,k,out} + V_{c,k,in})}} \tag{4.1}$$

The optimal value of k for a given c is $\arg\max_k T_{c,k}$. Assuming the LOF scores for a given c are random samples of Gaussian distribution $\mathcal{N}(\mu_{c,out}, \sigma^2_{c,out})$, $T_{c,k}$ approximates a non-central t distribution $T(k, \delta)$ with $k = 2\lfloor cn \rfloor - 2$ degrees of freedom and $\delta = \frac{\mu_{c,out} - \mu_{c,in}}{\sqrt{\frac{1}{\lfloor cn \rfloor}(\sigma^2_{c,out} + \sigma^2_{c,in})}}$ non-central parameter. To find the optimal value of c, standardized difference $T_{c,k_{c,opt}}$ cannot be compared directly as the distribution $T_{c,k}$ follows is different depending on c. Instead, quantiles for $T_{c,k_{c,opt}}$ in the corresponding distribution are compared. Optimal value of c is defined as $\arg\max_c P(Z < T_{c,k_{c,opt}}; df_c, ncp_c)$, where Z follows a non-central t distribution $T(df_c, ncp_c)$.

4.1.2 Contamination estimation

Python Outlier Detection Thresholding (`PyThresh`) [30] toolkit implements a variety of unsupervised non-parametric thresholding methods. The methods are designed to derive labels from outlier scores by estimating a threshold. They can be used to determine the contamination of a dataset. Some methods do not calculate the threshold value directly, but contamination can always be derived from the number of samples labeled as normal and anomalous. Thresholders implemented by `PyThresh` include:

- Z-score thresholder (ZSCORE). For mean $\bar{x}$ and standard deviation σ of outlier scores, Z-score is calulated as follows:
$$Z = \frac{x - \bar{x}}{\sigma} \tag{4.2}$$
Thresholder classifies any point with $|Z| > 1$ as an outlier.

- Trained Classifier thresholder (CLF). This thresholder estimates the threshold value using classifier trained with a linear stochastic gradient decent method. The model was trained with a warm start and fit with outlier scores and ground truth labels from PyOD outlier detection methods. The model uses scores, log of scores, and probability density function (PDF) of scores as inputs.

- Filtering based thresholder (FILTER). Outliers are determined by a Savitzky–Golay filter [47]. It is usually utilized for smoothing data and reducing the noise. In this

Algorithm 1 Automatic LOF parameter tuning

Require: X, Input data (n samples $\times$ d features)
Require: $grid_c$, Feasible contamination c values
Require: $grid_k$, Feasible neighborhood size k values

for all $c \in grid_c$ **do**
  for all $k \in grid_k$ **do**
    $M_{c,k,out} \leftarrow$ mean log LOF for $\lfloor cn \rfloor$ outliers
    $M_{c,k,in} \leftarrow$ mean log LOF for $\lfloor cn \rfloor$ inliers
    $V_{c,k,out} \leftarrow$ variance of log LOF for $\lfloor cn \rfloor$ outliers
    $V_{c,k,in} \leftarrow$ variance of log LOF for $\lfloor cn \rfloor$ inliers
    $T_{c,k} \leftarrow \frac{M_{c,k,out} - M_{c,k,in}}{\sqrt{\frac{1}{\lfloor cn \rfloor}(V_{c,k,out} + V_{c,k,in})}}$
  end for
  $M_{c,out} \leftarrow$ mean log $M_{c,k,out}$ over $k \in grid_k$
  $M_{c,in} \leftarrow$ mean log $M_{c,k,in}$ over $k \in grid_k$
  $V_{c,out} \leftarrow$ variance of $V_{c,k,out}$ over $k \in grid_k$
  $V_{c,in} \leftarrow$ variance of $V_{c,k,in}$ over $k \in grid_k$
  $ncp_c \leftarrow \frac{M_{c,out} - M_{c,in}}{\sqrt{\frac{1}{\lfloor cn \rfloor}(V_{c,out} + V_{c,in})}}$
  $df_c \leftarrow 2\lfloor cn \rfloor - 2$
  $k_{c,opt} \leftarrow \arg\max_k T_{c,k}$
end for
$c_{opt} \leftarrow \arg\max_c P(Z < T_{c,k_{c,opt}}; df_c, ncp_c)$

Ensure: c_{opt}, Optimal contamination value
Ensure: $k_{c,opt}$, Optimal neighborhood size

case, the filter is applied on the input scores and threshold is set to its maximum value.

- One-Class Support Vector Machine thresholder (OCSVM_T). Uses OcSVM to determine the outliers from outlier scores, specifically an Additive Chi2 kernel approximation and SGDOneClassSVM.
- Combined thresholder (COMB). Threshold is calculated as mean of specified thresholders, in this case ZSCORE, CLF, FILTER, OCSVM_T.

4.1.3 Overview of LOF application

In certain cases, LOF does not handle duplicated data well. Plot a in Figure 4.1 shows the outliers identified by LOF when data is not duplicated. In Plot B, there is duplicated data and the number of duplicates is shown above the points. Without duplicates, LOF was able to identify the outliers correctly. When duplicates are included, some points on the edges of a cluster were classified as anomalies. LOF is calculated as a ratio of density of a point to densities of its neighborhood. The area with highly duplicated samples is much denser than the outskirts of the cluster and this drastic change can manifest in a high outlier score, which then causes the points to be classified as outliers.

(a) LOF for data without duplicates

(b) LOF for data with duplicates

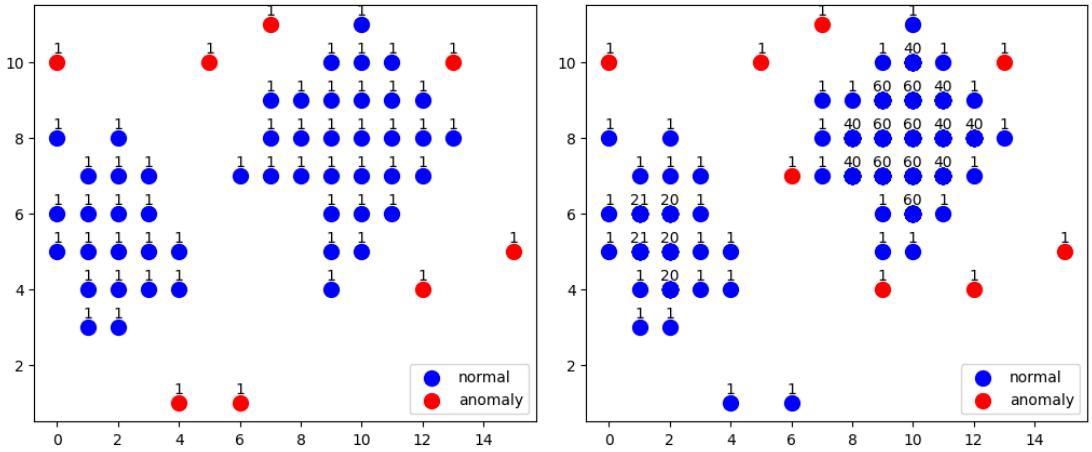

Figure 4.1: LOF outlier detection on dataset with duplicates vs. without duplicates. The the number of samples of a certain value is indicated by the number above the points.

To avoid this, after preprocessing the data, duplicates were removed before applying LOF. However, it is important to note that this can affect the anomalies, especially collective anomalies. Figure 4.2 displays the process of applying LOF algorithm on data. This pipeline was designed based on the capabilities of the parameter estimation methods and their evaluation, when applied on the LANL dataset, which are described in detail in Section 5.1. First, LOF scores are computed from the processed dataset for each candidate *MinPts*. The resulting scores are used to estimate a threshold with COMB thresholder. AT selects the optimal *MinPts* value for the calculated threshold, which is applied to calculated outlier scores for the selected *MinPts*, obtaining the final labeling.

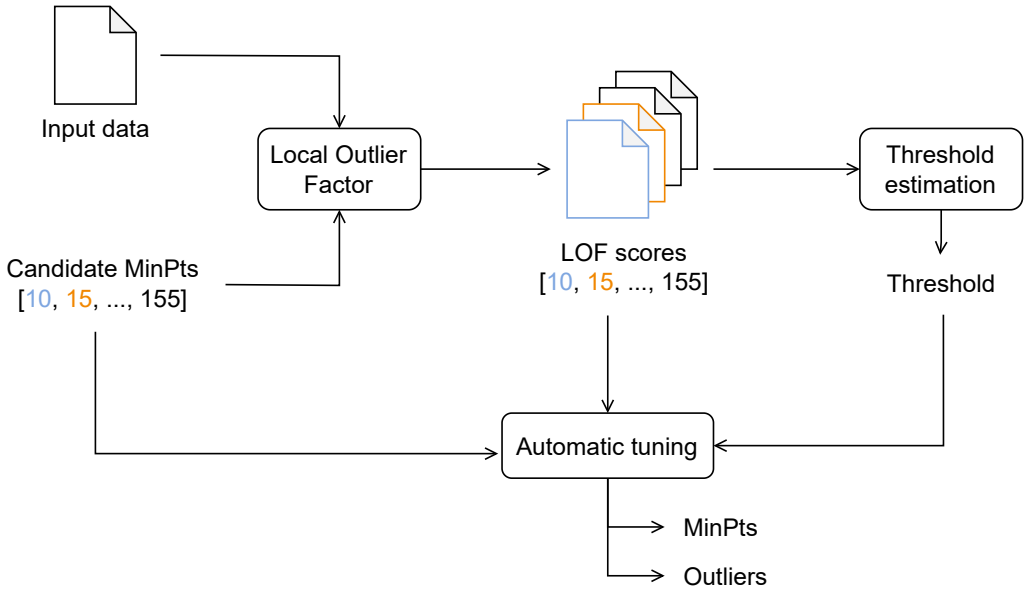

Figure 4.2: Overview of application of LOF on data.

4.2 DBSCAN

DBSCAN identifies clusters in data based on the density of their neighborhood. DBSCAN also recognizes the notion of noise, so it allows data points to not be assigned to any cluster. The results of clustering were interpreted for anomaly detection so that samples identified as noise (outliers) by DBSCAN were labeled as anomalies, while points belonging to a cluster were considered normal.

4.2.1 Parameter estimation

To apply DBSCAN, it was necessary to determine its parameters *Eps* and *MinPts*. *MinPts* was determined based on experiments with a variety of values, the process is detailed in Section 5.2. *Eps* was calculated using a method proposed by Alghamdi and Reger [3]. This method adopts the knee method to automatically calculate the parameter. Algorithm 2 outlines the calculation process.

Algorithm 2 *Eps* parameter estimation for DBSCAN

Require: X, Input data (n samples $\times$ d features)
Require: $MinPts$, minimum number of points in a neighborhood of a core point
drop duplicates in X
$kdist \leftarrow$ k-distances between data points calculated using KNN
$unique \leftarrow$ unique $kdist$
if $|unique| = 1$ **then**
    $Eps \leftarrow kdist_0$ / 2
else
    $Eps \leftarrow$ mean $unique$
end if
Ensure: *Eps*, Optimal *Eps* (neighborhood radius) value

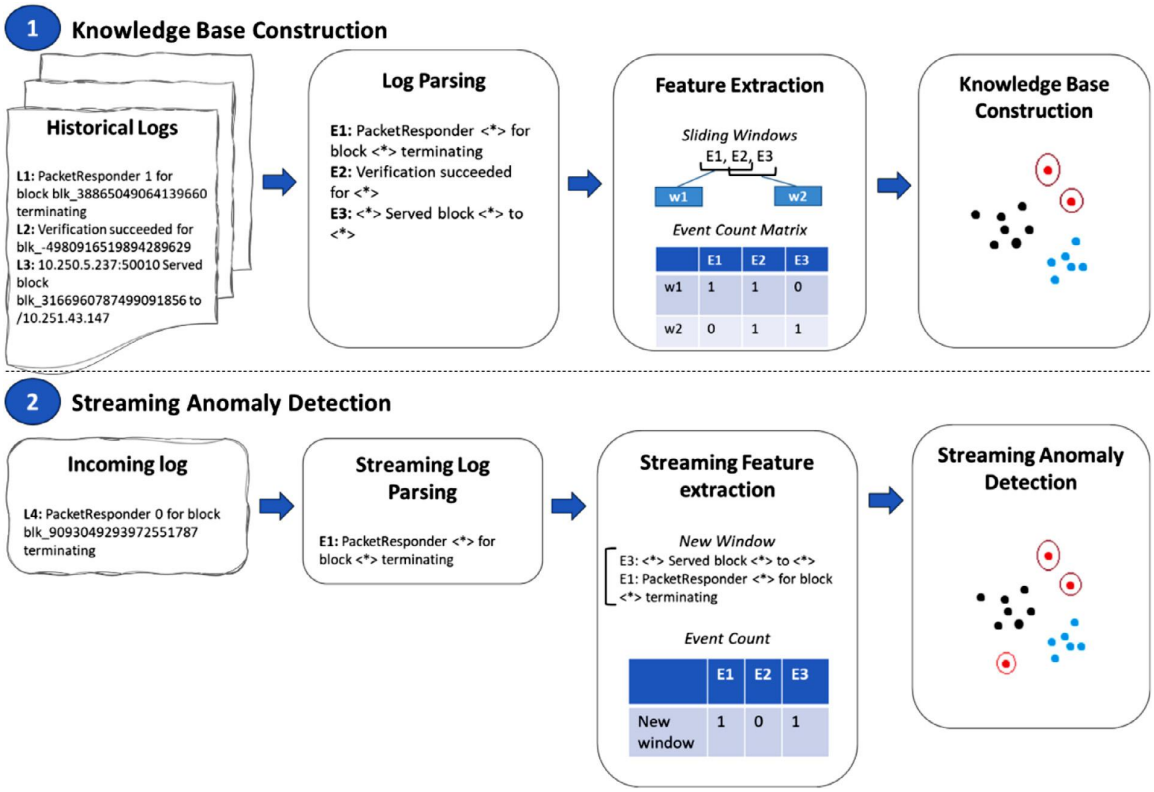

Figure 4.3: Overview of an OPTICS log anomaly detection framework. Adopted from [54].

4.3 OPTICS log anomaly detection framework

The next approach implemented in this work is an unsupervised anomaly detection framework proposed by Zeufack et al. [54]. This framework detects anomalies based on event occurrence patterns in logs. There are two phases: knowledge base construction and streaming anomaly detection. Figure 4.3 shows an overview of the framework.

Knowledge base construction is the training phase. During this stage, raw historical logs are processed and parsed into log templates (event types). Logs are partitioned by a sliding window of a fixed size k (number of samples in a window) that moves sequentially through the dataset with a specified stride, generating overlapping log sequences. Figure 4.4 shows an example partitioning with a sliding window of size three with a stride of one.

From each log sequence an event count vector feature is constructed to capture the event occurrence patterns. Size of the event count vector is equal to the number of templates extracted from logs. Each vector index corresponds to a certain template and its value is equal to the number of occurrences of that template in a given log sequence. For example, if the extracted templates are `{T0, T1, T2}`, for a log sequence `T2, T1, T1` the event count vector is `[0, 2, 1]`. The created event count vectors are clustered using OPTICS. The centroids of the extracted clusters form the knowledge base.

The second phase is the anomaly detection itself. The proposed framework detects anomalies in a streaming manner. For a new incoming log a sequence is formed with the $k-1$ previous logs. The sequence is converted into an event count vector and its distance to the closest centroid from the knowledge base is calculated. If the distance is

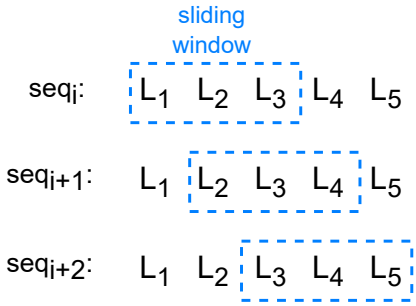

Figure 4.4: Partitioning of log samples (L_i) into sequences (seq_i) with a sliding window of size three and a stride of one.

greater than a threshold δ, the sequence is labeled as anomalous. Threshold δ is the largest distance between a centroid and a member of its cluster, across all clusters.

4.3.1 Parameter estimation

The *Eps* parameter influences how many clustering levels (different cluster densities) are detected. It is usually set to the maximum value (such as `numpy.inf` in `python`). However, OPTICS is quite insensitive to the *Eps* parameter [6]. In practice, values lower than the maximum can be used with little impact on the clustering results and can lead to a decreased runtime [48].

There is no definitive heuristic for choosing the $MinPts$ value. Schubert and Getrz [48] suggest that guidelines similar to the DBSCAN parameter may apply, referring Sander et al. [46], who recommend using a value of at least twice the dataset dimensions. They also recommended to increase the value for datasets with large amount of noise or duplicates. The experiments were conducted with a range of $MinPts$ values, based on these recommendations.

4.4 Implementation tools

The selected methods were implemented in `python`. The following summarized utilized libraries and toolkits and their usage:

- `pandas`[1], `numpy`[2] – used for data manipulation and preprocessing;
- `scikit-learn`[3] – used for implementation of anomaly detection algorithms and evaluation;
- `PyOD` [55] (Python Outlier Detection toolkit) – used for LOF implementation;

[1]Pandas: Python Data Analysis Library. Available at: https://pandas.pydata.org/

[2]NumPy: the fundamental package for scientific computing with Python. Available at: https://numpy.org/

[3]Scikit-learn: Machine Learning in Python. Available at: https://scikit-learn.org/stable/

- `PyThresh`[4] (Python toolkit for thresholding outlier detection likelihood scores) – used for implementing LOF contamination estimation;
- DBSCAN-based framework by Alghamdi and Reger [3] – this work uses parts of the implementation modified for the implemented task, namely the *Eps* calculation, evaluation and exporting of clustering results;
- `drain3`[5] – this implementation of the Drain parser is used for log parsing and template extraction.

[4]PyThresh: Python toolkit for thresholding outlier detection likelihood scores. Available at: https://pythresh.readthedocs.io/en/latest/?badge=latest

[5]Drain3: Online log template miner. Available at: https://github.com/logpai/Drain3

Chapter 5

Evaluation

This chapter contains evaluation of the implemented methods. Each selected approach was firstly applied on a labeled dataset. If multiple techniques for parameter estimation were considered, the one with the best F1 score achieved on the labeled dataset was chosen as the final approach. The effectiveness of the method as a whole was estimated and the method was applied to the AT&T dataset.

AT&T dataset is unlabeled so it was not possible to compare the predicted labels to a ground truth when evaluating the outputs. Generally, to confirm whether flagged samples are true anomalies, in the context of unsupervised learning, domain experts have to review the results manually. An anomaly can be verified by linking it to a known security incident that occurred, based on further investigation of history of a suspicious user or in other ways. This kind of background information was not provided for the AT&T dataset. Therefore, the effectiveness of the methods cannot be estimated precisely. However, exploration of the results still revealed some interesting observations.

One of the goals of this work was to select a method with the lowest rate of false alarms (false positives, while also detecting as many of the present anomalies as possible. These characteristics are expressed in the F1 score, so for the purpose of this work, the methods were be compared based on this metric. Therefore, hereafter, when a solution is called more "efficient" it refers to achieving a higher F1 score.

5.1 Local Outlier Factor

The efficiency of LOF and the selected parameter estimation methods were first evaluated on the LANL dataset before being applied to the AT&T dataset.

5.1.1 LANL dataset

Before applying LOF, all duplicated data points were removed, as detailed in Section 4.1.3. Table 5.1 shows the original number of samples and the number of samples remaining after the duplicates were removed. The table includes counts for normal and all anomalous data, along with counts for each anomaly type. As expected, collective anomalies which are user specific, such as `BurstLogonUser` and `ManyFailsUser` contained a large number of duplicates.

Methods chosen for parameter estimation were evaluated, starting with the tuning method AT. Candidate *MinPts* values were set to a range from ten to 150 with a step of five, $grid_k = \{10, 15, 20, 25, ..., 155\}$, to represent a variety of values. Removing dupli-

Table 5.1: Number of normal and anomalous events in LANL dataset before and after removing duplicate samples. Anomaly type `all` is the total sum of samples for all anomaly types.

Label	Anomaly Type	Original	Without Duplicates	Removed
Normal	-	944,421	243,913	700,508
Anomaly	all	19,054	18,176	878
Anomaly	UnusualActivity	1919	1911	8
Anomaly	UnusualActivityUser	1923	1902	21
Anomaly	BurstLogonAttrib	1915	1909	6
Anomaly	BurstLogonUser	1911	1684	227
Anomaly	ManyFailsGlobalUnspec	1907	1906	1
Anomaly	ManyFailsGlobal	1903	1898	5
Anomaly	ManyFailsUser	1900	1717	183
Anomaly	UnexAuthUser	1896	1706	190
Anomaly	NontypAuthUser	1892	1660	232
Anomaly	NontypAuth	1888	1883	5

cates discarded some anomalies, but most removed records were normal, which changed the contamination of the resulting data. The original contamination was 2% and the contamination of dataset without duplicates was approximately 7.5%. Therefore, the candidate contamination values had to be set appropriately higher than would normally be expected (typical contamination is 1-3%). The selected candidate values for contamination were $grid_c = \{0.02, 0.04, 0.06, 0.08, 0.1\}$.

The results of AT parameter tuning are $k_{opt} = 70, c_{opt} = 0.2$. However, these parameters are far from optimal. In fact, out of all the $grid_c$ values, LOF with a 2% contamination performed the worst, which can be seen in Table 5.2. The table shows evaluation of LOF with different values of contamination and their respective optimal *MinPts*. AT selects the contamination value which corresponds to the highest quantile for a non-central t distribution. For all $c \in grid_c$, the computed quantile was 1.0, so AT considered all values as equally optimal. The resulting value was chosen based on the implementation of arg $\max_c$, which picked the first argument out of multiple equal options, producing incorrect results.

AT can also be used to only tune the *MinPts* parameter if contamination is known. Table 5.2 shows performance of LOF with *MinPts* calculated by AT for a given fixed contamination, compared to the best results for the same contamination level. AT selected the optimal MinPts value in all cases except for 2% contamination, where the difference was negligible.

In conclusion, while AT did not produce meaningful results for joint tuning of both parameters, it could be used to find a nearly optimal value of *MinPts* for a given contamination. Contamination had to be determined in another way. Five different thresholding methods were evaluated: ZSCORE, FILTER, OCSVM_T, CLF, and COMB, which combined the thresholds calculated by all other methods by averaging. The thresholding methods can compute thresholds for multiple sets of outlier scores; therefore, thresholds were calculated from LOF scores for all candidate *MinPts* values. The resulting threshold c was used as an input for AT to obtain a *MinPts* value k. Table 5.3 shows efficiency of LOF with parameters c and k. The combination of all estimators, COMB, achieved the highest F1 Score, so this method was selected.

Table 5.2: Comparison of LOF performance with *MinPts* calculated by Automatic tuning and the highest achieved performance for any $MinPts \in grid_k$, with respect to a fixed contamination (C).

C	Automatic tuning				Best			
	MinPts	Precision	Recall	F1 Score	MinPts	Precision	Recall	F1 Score
2%	70	0.5661	0.1632	0.2534	50	0.5850	0.1687	0.2619
4%	70	0.4732	0.2729	0.3462	70	0.4732	0.2729	0.3462
6%	70	0.3902	0.3376	0.3620	70	0.3902	0.3376	0.3620
8%	70	0.3356	0.3872	0.3595	70	0.3356	0.3872	0.3595
10%	70	0.2991	0.4313	0.3532	70	0.2991	0.4313	0.3532

Table 5.3: Efficiency of LOF using different thresholding estimation methods. C stands for contamination of dataset. $MinPts$ values were estimated with AT method.

Thresh. estimator	C	MinPts	Precision	Recall	F1 Score
ZSCORE	4.09%	70	0.4685	0.2767	0.3480
FILTER	1.86%	70	0.5656	0.1519	0.2395
OCSVM	10.39%	70	0.2935	0.4400	0.3521
CLF	10.25%	70	0.2956	0.4373	0.3528
COMB	6.65%	70	0.3687	0.3537	**0.3611**

To summarize, the most efficient method to estimate the dataset contamination is a COMB thresholder and AT can select a nearly optimal value of $MinPts$. The parameters estimated for LANL dataset using these methods are $MinPts = 70$ and $contamination = 6.653\%$. LOF anomaly detection applied to LANL with these parameters achieved precision 0.36, recall 0.35 and F1 score 0.36.

5.1.2 AT&T dataset

Removing duplicates reduced the AT&T dataset much more drastically than LANL. From the initial 865,619 samples, only 23,573 remained. It seems that the users whose activity was recorded on the server, repeatedly performed the same actions and because the timestamp is not included as one of the features, these actions produced samples with identical values (after preprocessing). Using the same parameter estimation process as for the LANL dataset, the resulting parameter values were $MinPts = 70$ and $contamination = 6.198\%$. In total, LOF identified 1,461 samples as anomalies.

Analyzing the anomalous samples on attribute level revealed that certain attribute values only occurred in anomalous samples. This could be a strong indicator that the attribute value was the cause of the anomalousness or that the value was common for a group of samples that deviated from normal. One such example was the customer name `Customer 1`, which was one of the two `customer_name` values in the dataset. All samples of `Customer 1` were identified as anomalies. This customer value was much less frequent, only 61 samples out of the total 20 thousand. It could be simply this fact that made it anomalous; however, on further investigation, it shows that this subset of samples was unique in more ways. `Customer 1` was always connected to the same destination IP address and the address did not appear for the other customer. The commands that were connected to this customer were generally very infrequent, with respect to the whole dataset. Furthermore, the users

Table 5.4: DBSCAN performance on LANL dataset for various *MinPts* values. Highest achieved F1 score is in bold.

MinPts	Eps	Precision	Recall	F1 Score
2	56.92	0.4681	0.5242	**0.4945**
3	54.06	0.3929	0.6035	0.4759
5	59.05	0.3324	0.6693	0.4442
10	67.46	0.2684	0.7442	0.3945
15	74.97	0.2215	0.8292	0.3497
20	88.84	0.1861	0.8834	0.3075
30	104.51	0.1518	0.9464	0.2617

connected to `Customer 1` rarely appeared for the other customer. All of this seems to indicate that samples with `Customer 1` exhibited different behavior patterns and there was not enough data for this customer to consider these patterns normal. It also showed that LOF was able to identify these deviating patterns even when they were connected to a group of samples (not just individual instances).

Another example was the source IP address `0.0.1.228` which was always anomalous (only occurred in anomalous samples). The IP address was used quite frequently and in connection with multiple users. That is to say, the cause for anomalousness was not a low global frequency. Instead, it seems that the IP address was determined abnormal with respect to specific users. Users with the source IP address `0.0.1.228` were also connected to other source addresses and out of the multiple values, `0.0.1.228` was consistently considerably less frequent. So, it seems that it was identified as abnormal with respect to a given user, showing that the method can identify patterns with respect to attributes (not just on the global level).

5.2 DBSCAN

DBSCAN efficiency was estimated using the LANL dataset. The effect of different $MinPts$ values on the precision, recall and F1 score was observed and the most appropriate value was selected. DBSCAN was then applied to the AT&T dataset.

5.2.1 LANL dataset

Table 2 displays the performance of DBSCAN on LANL dataset for a range of *MinPts* values. The *Eps* was calculated with respect to *MinPts* using method outlined in Section 4.2.1. With increasing *MinPts*, the recall increased but the precision lowered, which means that while more true anomalies were identified as anomalous, more normal data points were as well.

Naturally, smaller *MinPts* produces smaller clusters, which can lead to anomalies forming clusters and therefore to classifying them as normal. For example, for $MinPts = 2$, out of the 8,898 total clusters formed by DBSCAN, 1,077 of them were purely anomalous, that is, they only contained anomalous data samples, and 189 contained both anomalies and normal data. Table 5.5 shows how many anomalies of each type were identified as an outlier by DBSCAN and how many were a member of cluster. Among the types, `BurstLogonUser`, `ManyFailsUser` were misclassified the most. This is understandable, as they are collec-

Table 5.5: Overview of DBSCAN clustering results for each anomaly type present in LANL dataset.

Anomaly type	Outlier	In a cluster
BurstLogonAttrib	1,481	434
BurstLogonUser	307	1,604
ManyFailsGlobal	1,380	523
ManyFailsGlobalUnspec	1,742	165
ManyFailsUser	316	1,584
NontypAuth	1,478	410
NontypAuthUser	300	1,592
UnexAuthUser	312	1,584
UnusualActivity	1,521	398
UnusualActivityUser	152	771
Total	**9989**	**9065**

tive anomalies. These are all also anomaly types tied to a specific user. Interestingly, `manyFailsGlobal` was detected much more successfully than `manyFailsUser`, even though both are collective anomalies of the same nature. This is because as the samples share the same user, they were naturally more similar and could, therefore, more easily form a cluster.

DBSCAN achieved the highest F1 score for $MinPts$ value of two. The bigger the $MinPts$ value, the less effective was the anomaly detection. This is connected to how DBSCAN was applied; anomalies were equated with noise (as defined in the context of DBSCAN), so the algorithm identified the most anomalous samples in a global context of the whole dataset. In line with this usage, the smallest value of $MinPts$ was the most suitable, since it allowed samples of an infrequent behavior to form clusters as well, reducing the number of false positives.

To summarize, the selected value of $MinPts$ was two. The Eps value calculated for the LANL dataset, with respect to $MinPts$ was $Eps = 56.92$. DBSCAN anomaly detection applied to LANL dataset with these parameters achieved precision 0.45, recall 0.52 and an overall F1 score 0.49.

5.2.2 AT&T dataset

DBSCAN was applied with parameters $MinPts = 2$ and $Eps = 0.0613$, producing 434 clusters and 99 outliers. The number was much lower when compared to results of LOF, especially when taking into account that DBSCAN was applied to the full dataset (as apposed to removing duplicates). This was caused by the $MinPts$ parameter, that specified that only two points are required to form a cluster. This means that samples classified as anomalies by LOF, for example samples connected to `Customer 1`, formed their own cluster and therefore were not labeled as outliers.

The results contained almost no attribute values that would occur only in anomalous samples. Instead, the algorithm identified the most abnormal value combinations with respect to the whole dataset. This made it especially challenging to evaluate, since the anomalousness depended on a combination of attributes which was not easily visible.

However, there were also some immediately suspicious behaviors. For example, there were three users whose logins were identified as anomalous. These users performed login

on two separate days, each day all three logins were from the same source IP address (the address was different each day) and the logins happened at the same time. After the login, users did not perform any other actions. To draw any definite conclusions, such as that an attacker trying to gain access to user accounts was responsible for this activity, it would be necessary to examine the history of the users or verify their actions in person, which was not possible within the scope of this work.

Another suspicious behavior was identified for user `mm930m`. Usually, the user performs multiple actions with respect to a destination IP address. The identified samples document unusual behavior, where the user only performed one command of type `TACACS+_PPP` (communication with a point-to-point protocol) for a particular destination IP address. Such an isolated event could be considered suspicious. Again, to confirm whether it was truly a malicious action, more information would be necessary, for example examining the contents of the communication. Nonetheless, in both presented examples, DBSCAN appeared to successfully identify certain suspicious behavior that was abnormal, with respect to a combination of attributes.

5.3 OPTICS log anomaly detection framework

The OPTICS framework was evaluated using the HDFS dataset. Since the original paper lacked details on parameter settings, implementation, and other specifics, the framework was initially applied to the HDFS dataset to verify the reproducibility of the results. Afterwards, the framework was applied on AT&T dataset.

5.3.1 HDFS evaluation

Table 5.6 shows the performance using different $MinPts$ values for the OPTICS parameter. As mentioned in Section 4.3.1, it is recommended that $MinPts$ is at least twice the number of dataset attributes and even higher if the dataset contains large amount of duplicates. In this case, the OPTICS input was an event count matrix with approximately 350 thousand rows, of which only around 70 thousand were unique. There were 33 detected templates for HDFS, meaning the event count vector had 33 dimensions. Based on these characteristics, a wider range of $MinPts$ values from 70 up was tested. Eps was set to `numpy.inf` for all experiments. The knowledge base was calculated using 70% of data and 30% was used for testing. The framework performed best for $MinPts = 150$. It can also be noted, that at a particular $MinPts$ value, the precision remained relatively consistent for any value above it.

Table 5.6: Efficiency of OPTICS framework for a variety of $MinPts$ values.

MinPts	Precision	Recall	F1 Score
50	0.9269	0.0345	0.0665
100	0.6970	0.2920	0.4116
120	0.7139	0.5784	0.6391
150	0.7141	0.6052	**0.6551**
200	0.7154	0.4702	0.5674

5.3.2 AT&T results

There were 218 templates extracted from the AT&T dataset. Using all templates would have lead to a very large event count matrix and long runtime of OPTICS. Instead, only the 50 most frequently occurring templates were considered. An extra column was added to record the sum of counts of the less frequent templates. The event count matrix contained approximately 600 thousand rows and around 420 thousand of them were unique. Considering the proportion of duplicates was lower than for the HDFS dataset but the count vectors had more dimensions, it was assumed that $MinPts$ value that performed well for HDFS would also produce reasonable results for AT&T dataset. Therefore, the OPTICS was applied with $MinPts = 150$ and $Eps =$ `numpy.inf`. The knowledge base was constructed from 70% of AT&T data and the streaming anomaly detection was applied to the remaining 30%. Of the 250 thousand analyzed windows, 85 were identified as anomalous.

5.4 Summary and comparison with similar approaches

In summary, three methods were applied on the selected datasets: LOF, DBSCAN and an OPTICS framework. Table 5.7 compares their results. Out of the methods evaluated on the LANL dataset, DBSCAN achieved the highest F1 score of 0.49. Overall, OPTICS framework attained the best results with an F1 score of 0.65.

When comparing the anomalies detected in the AT&T dataset by all three methods, it was found that 75% of the anomalies detected by DBSCAN were also identified by LOF. However, the log sequences identified as anomalous by the OPTICS framework did not correspond to anomalies detected by either LOF or DBSCAN. This outcome is not entirely unexpected. While LOF and DBSCAN mainly detect point anomalies, the OPTICS framework identifies collective anomalies at a log sequence level. This, combined with a shortened event count vector, resulted in point anomalies that went undetected by the OPTICS framework.

Table 5.7: Comparison of all selected solutions evaluated on labeled datasets.

Method	Dataset	Precision	Recall	F1 Score
LOF	LANL	0.3687	0.3537	0.361
DBSCAN	LANL	0.4681	0.5242	0.4945
OPTICS FW	HDFS	0.7141	0.6052	**0.6551**

To contextualize the results, several works that implemented similar approaches were examined for comparison. The implemented OPTICS framework achieved the same precision as in the original paper. Nonetheless, Zeufack et al. [54] achieved a higher recall of 1.0 and the overall F1 score of 0.83. This difference may be largely due to the variation in dataset size, as their study applied the framework to three million HDFS samples, while this work used only 500 thousand samples. Björnerud [8] presented a similar approach but used DBSCAN instead of OPTICS for clustering the event count vectors, achieving an F1 score of 0.6%. Compared to this, the OPTICS framework performed slightly better. However, Björnerud also applies DBSCAN on the same data after removing duplicated sequences, which improved the F1 score to 0.95. This presents an interesting area for future research.

Šiklóši [57] applies LOF a network traffic dataset and generated syslog datasets of varying size. LOF achieved an F1 score of 0.048 on the traffic dataset. The performance of LOF on the syslog data was largely depended on the size of dataset, an F1 score of 0.45

was achieved on just five thousand samples, however, the performance decreased to zero when LOF was applied to dataset with 50 thousand log messages.

It was challenging to find works that applied DBSCAN in a similar manner to this thesis. Most studies use DBSCAN more alike the OPTICS framework or combined with other anomaly detection techniques. However, Kommineni [28] included DBSCAN as one of the baseline models applied to logs in the same way as this work, attaining an F1 score of 0.5. In conclusion, the achieved results are comparable to the results of the selected works that adopted a similar approach.

Chapter 6

Application on large real-world log files

The need to analyze large amounts of data, which are unmanageable manually, is one of the driving forces for using machine learning for log anomaly detection. Therefore, it is important that the used methods are suitable for handling large real-world log files.

The "suitability" of a method can be considered in several aspects, out of which the following three are discussed: the theoretical characteristics of an algorithm, implementation factors, and method application. The first two factors examine the effort of a one-time application of anomaly detection. Theoretical characteristics include the time and space complexity of the utilized algorithm and implementation factors examine the parallelization options. The third aspect, method application, takes into account efforts necessary for a continued use of the method. Logs are inherently data streams and new data are generated continuously which influences the usability of certain methods.

6.1 LOF

The LOF algorithm consists of two main steps. The first step is finding $MinPts$-nearest neighbors for all data points. For dataset of size n, a K-nearest neighbor (KNN) query has to be performed n times leading to time complexity of $\mathcal{O}(n * \texttt{knn})$, where `knn`, for KNN query complexity `knn`. A naive KNN approach calculates the distance to every data point for all points, resulting in a complexity of $\mathcal{O}(n)$ [6]. If the query is computed using special data structures such as k-d trees or ball trees, the complexity can be reduced to $mathcal O(\log n)$. However, for highly dimensional data, the performance may degrade and be surpassed by the naive approach [51]. Thus, the first LOF step has a time complexity of $\mathcal{O}(n \log n)$ for low and medium dimensional dataset and complexity of $\mathcal{O}(n^2)$ for highly dimensional datasets. The second step is computing the LOF scores. This phase requires iterating over the $MinPts$ neighbors of each point leading to complexity of $\mathcal{O}(n)$. Apart from all the data points, the algorithm needs to store calculated distances, which leads to space complexity $\mathcal{O}(n)$.

From an implementation stand point, the algorithm is easily parallelizable. The calculation of nearest neighbors, the subsequent local reachability density (lrd) and of the LOF scores are independent for each point.

LOF is intended for use on a static dataset. To account for new generated logs, anomaly detection process must be repeated. If previous logs are included in subsequent anomaly

detection runs, the dataset quickly increases to an unmanageable size. However, if only a chunk of data (such as logs from the last day) is analyzed, the results may be skewed because short time periods do not provide a robust representation of normal behavior. Regardless, the calculation of LOF has to be repeated for all data points each run. These drawbacks were the motivation for development of several LOF modifications specifically suited for data streams. Alghushairy et al. [4] provide a review of the ILOF, MILOF and DILOF modifications. When a new data point p is added, ILOF calculates KNN, lrd and LOF score for the new point and updates points around p, whose scores are affected by the addition. This way, efforst needed for analyzing new data are reduced. ILOF is still demanding memory-wise, since the previous data points need to be stored in memory, which also slows down the calculations. MILOF and DILOF try to solve the issues by storing the past points in a summarized form, by using k-means clustering and gradient descent, respectively.

6.2 DBSCAN

Similarly to LOF, DBSCAN performs a KNN query for each data point. If the query is performed using optimized indexing structures (k-d tree or ball tree), the overall complexity reaches $\mathcal{O}(n \log n)$. For cases, where the structures cannot be used, such as for highly dimensional data, the complexity with a naive KNN query is $\mathcal{O}(n^2)$. To avoid repeated calculation of distances between data points, DBSCAN can store pairwise distances in a matrix, which yields the space complexity of $\mathcal{O}(n^2)$. Implementations that do not utilize the matrix have space complexity of $\mathcal{O}(n)$.

DBSCAN algorithm can be parallelized. KNN queries can be computed independently and the expansions of clusters can be performed simultaneously. The original version of DBSCAN operates on a static dataset which causes the sames issues as outlined in Section 6.1. Silva et al. [49] present a survey of data stream clustering techniques, including approaches based on DBSCAN, such as ClusTree [29], D-Stream [13], and DenStream [11].

6.3 OPTICS log anomaly detection framework

OPTICS can be considered an extension of DBSCAN, therefore, it shares the same characteristics when it comes to the algorithm complexity. The time complexity is governed by the KNN query for each data point and is overall $\mathcal{O}(n \log n)$ if indexing structures are used. The worst case still reaches $\mathcal{O}(n^2)$. The authors also determined that on average OPTICS is 1.6 times slower than DBSCAN [6]. The space complexity can reach up to $\mathcal{O}(n^2)$ if a distance matrix is used to store pairwise point distances, which is generally avoided in practical settings. Otherwise, the space complexity is $\mathcal{O}(n)$.

OPTICS is inherently difficult to parallelize. As the output is determined by the processing order, the individual KNN queries and the subsequent distance calculations have to be performed sequentially. The framework was designed for online anomaly detection, which means the training (knowledge base construction) is performed once in the beginning and then new data points can be evaluated with minimal effort. However, it may be necessary to update the knowledge base from time to time to account for new data patterns and new log templates that inevitably appear in logs of evolving systems.

6.4 Summary

Table 6.1 summarizes characteristics of the used methods with regards to processing large log files. From a theoretical standpoint, all algorithms are comparable. When considering the real-world use on log data of a streaming character, the OPTICS framework is the best solution. It requires performing the OPTICS clustering only during the training phase and the anomaly detection of new data requires an insignificant effort when compared to the other methods. However, there are several works that adopt DBSCAN and LOF for a streaming detection. If instead, the modified methods were used, combined with the possibility of parallelized computation, they might represent a better solution. Exploring LOF and DBSCAN in a streaming setting presents a topic for future work.

Table 6.1: Characteristics of selected methods with regards to time and space complexity and application on large log data files.

Method	Time complexity		Space complexity		Parallelizable	Suitable for data streams
	Worst	Normally	Worst	Normally		
LOF	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	✓	✗
DBSCAN	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	✓	✗
OPTICS FW	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	✗	✓

Chapter 7

Conclusion

This work focused on log anomaly detection with machine learning. The goal was to select a method applicable to dataset provided by AT&T, which contains logs from an AAA server. The AT&T dataset is not annotated, therefore, unsupervised approaches were used. The selected methods were: LOF, DBSCAN and an OPTICS-based anomaly detection framework. To evaluate the efficiency of the methods, two labeled datasets were selected for validation. The AT&T dataset contains a mix of categorical attributes an unstructured text, the methods process one or the other. The HDFS dataset with unstructured log messages was selected for validation of the OPTICS framework. LANL dataset, enriched with generated anomalies, was used for validation of LOF and DBSCAN. Overall, the OPTICS framework achieved the highest efficiency with a F1 score of 0.65. Out of the methods applied to the LANL dataset, DBSCAN proved to be more effective attaining a F1 score of 0.49. The results are comparable to similar works.

The methods were applied on the AT&T dataset and successfully identified suspicious behavior that deviates from normal patterns. However, to confirm whether the identified samples are true anomalies, additional knowledge, such as user history, is required. At the time of creation of this work, AT&T was not able to provide such information.

7.1 Future work

In future work, after more information and domain knowledge is obtained, the application of methods to the AT&T dataset could be evaluated more specifically. This would enable determining the most suitable method for this distinct dataset. Additionally, evaluating the methods on multiple datasets could help identify a more robust method. Currently, ten types of anomalies were generated for the LANL dataset; future work could expand this number and include more elaborate attack scenarios to provide a more representative dataset for evaluating the methods.

Other areas with a potential for improvement involve the selected methods. Both LOF and DBSCAN were applied in a static setting to the entirety of the dataset. In a real-world scenarios, anomaly detection would have to be performed periodically, requiring a substantial amount of resources for computation. However, there are several works that adopt the algorithms for data streams, which limits the efforts necessary for analyzing new data. The use of these variants could greatly improve the practical applicability of the methods.

In this work, DBSCAN was applied by equating anomalies with noise identified by DBSCAN. It could be beneficial to explore other applications, such as identifying anomalies based on cluster size or the distance from a cluster centroid. The OPTICS framework, as adopted in this work, does not classify individual samples but rather whole sequences. Authors of the framework also do not specify how to identify anomalies on a sample level. Nonetheless, extending the framework in this way would be valuable, leading to more easily interpretable results.

Bibliography

[1] Abdi, H. and Valentin, D. Multiple correspondence analysis. *Encyclopedia of measurement and statistics*. Thousand Oaks (CA): Sage. 2007, vol. 2, no. 4, p. 651–657.

[2] Aksoy, S. and Haralick, R. M. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern recognition letters*. Amsterdam, Netherlands: Elsevier. 2001, vol. 22, no. 5, p. 563–582.

[3] Alghamdi, A. A. and Reger, G. Pattern extraction for behaviours of multi-stage threats via unsupervised learning. In: IEEE. *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. 2020, p. 1–8.

[4] Alghushairy, O., Alsini, R., Soule, T. and Ma, X. A review of local outlier factor algorithms for outlier detection in big data streams. *Big Data and Cognitive Computing*. Basel, Switzerland: MDPI. 2020, vol. 5, no. 1, p. 1.

[5] Ali, A., Shamsuddin, S. M. and Ralescu, A. L. Classification with class imbalance problem. *Int. J. Advance Soft Compu. Appl*. Jordan: Al-Zaytoonah University of Jordan. 2013, vol. 5, no. 3, p. 176–204.

[6] Ankerst, M., Breunig, M. M., Kriegel, H.-P. and Sander, J. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record*. NY, USA: ACM New York. 1999, vol. 28, no. 2, p. 49–60.

[7] Astekin, M., Zengin, H. and Sözer, H. DILAF: A framework for distributed analysis of large-scale system logs for anomaly detection. *Software: Practice and Experience*. Wiley Online Library. 2019, vol. 49, no. 2, p. 153–170.

[8] Björnerud, P. *Anomaly detection in log files using machine learning*. Luleå, Sweden, 2021. Master's thesis. Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering.

[9] Breunig, M. M., Kriegel, H.-P., Ng, R. T. and Sander, J. LOF: identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, p. 93–104.

[10] Brownlee, J. *Ordinal and one-hot encodings for categorical data* [online]. August 2020 [cit. 2024-03-25]. Available at: https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/.

[11] Cao, F., Estert, M., Qian, W. and Zhou, A. Density-based clustering over an evolving data stream with noise. In: *Proceedings of the 2006 SIAM international conference on data mining*. Philadelphia, USA: SIAM, 2006, p. 328–339.

[12] Chandola, V., Banerjee, A. and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*. NY, USA: ACM New York. 2009, vol. 41, no. 3, p. 1–58.

[13] Chen, Y. and Tu, L. Density-based clustering for real-time stream data. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007, p. 133–142.

[14] Cheng, Z., Zou, C. and Dong, J. Outlier detection using isolation forest and local outlier factor. In: *Proceedings of the conference on research in adaptive and convergent systems*. 2019, p. 161–168.

[15] Dalatu, P. and Midi, H. New approaches to normalization techniques to enhance K-means clustering algorithm. *Malaysian Journal of Mathematical Sciences*. Universiti Putra Malaysia. 2020, vol. 14, no. 1, p. 41–62. ISSN 1823-8343.

[16] Dani, M. C., Doreau, H. and Alt, S. K-means application for anomaly detection and log classification in hpc. In: Springer. *Advances in Artificial Intelligence: From Theory to Practice: 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part II 30*. New York, NY, USA: [b.n.], 2017, p. 201–210.

[17] Dridi, S. Supervised learning-a systematic literature review. OSF Preprints. 2021.

[18] Ester, M., Kriegel, H.-P., Sander, J., Xu, X. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis, E., Han, J. and Fayyad, U. M., ed. *KDD*. Washington D.C, USA: AAAI Press, 1996, vol. 96, no. 34, p. 226–231. ISBN 1-57735-004-9.

[19] Farshchi, M., Schneider, J.-G., Weber, I. and Grundy, J. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In: IEEE. *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*. 2015, p. 24–34.

[20] Goldstein, M. and Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*. Public Library of Science San Francisco, CA USA. 2016, vol. 11, no. 4, p. e0152173.

[21] Han, J., Kamber, M. and Pei, J. 3 - Data Preprocessing. In: Han, J., Kamber, M. and Pei, J., ed. *Data Mining (Third Edition)*. Third Editionth ed. Boston: Morgan Kaufmann, 2012, p. 83–124. The Morgan Kaufmann Series in Data Management Systems. ISBN 978-0-12-381479-1.

[22] He, P., Zhu, J., Zheng, Z. and Lyu, M. R. Drain: An online log parsing approach with fixed depth tree. In: IEEE. *2017 IEEE international conference on web services (ICWS)*. 2017, p. 33–40.

[23] He, S., Zhu, J., He, P. and Lyu, M. R. Experience report: System log analysis for anomaly detection. In: *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, p. 207–218. DOI: 10.1109/ISSRE.2016.21.

[24] Henriques, J., Caldeira, F., Cruz, T. and Simões, P. Combining k-means and xgboost models for anomaly detection using log datasets. *Electronics*. Basel, Switzerland: MDPI. 2020, vol. 9, no. 7, p. 1164.

[25] Jain, A. K., Murty, M. N. and Flynn, P. J. Data clustering: a review. *ACM computing surveys (CSUR)*. NY, USA: ACM New York. 1999, vol. 31, no. 3, p. 264–323.

[26] Japkowicz, N. and Stephen, S. The class imbalance problem: A systematic study. *Intelligent data analysis*. Amsterdam, Netherlands: IOS Press. 2002, vol. 6, no. 5, p. 429–449.

[27] Kent, A. D. *Comprehensive, Multi-Source Cyber-Security Events* [Los Alamos National Laboratory]. 2015. DOI: 10.17021/1179829.

[28] Kommineni, S. S. M. *Automating Log Analysis*. Karlskrona, Sweden, 2021. Master's thesis. Blekinge Institute of Technology, Faculty of Computing, Department of Computer Science.

[29] Kranen, P., Assent, I., Baldauf, C. and Seidl, T. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems*. New York, NY, USA: Springer. 2011, vol. 29, p. 249–272.

[30] Kulik, D., Schmidl, S. and Perini, L. *KulikDM/pythresh: v0.3.6*. Zenodo, feb 2024. DOI: 10.5281/zenodo.10613189. Available at: https://doi.org/10.5281/zenodo.10613189.

[31] Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y. and Chen, X. Log clustering based problem identification for online service systems. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA: Association for Computing Machinery, 2016, p. 102–111. ICSE '16.

[32] Longadge, R. and Dongre, S. Class imbalance problem in data mining review. *ArXiv preprint arXiv:1305.1707*. 2013.

[33] Lv, D., Luktarhan, N. and Chen, Y. ConAnomaly: Content-based anomaly detection for system logs. *Sensors*. Basel, Switzerland: MDPI. 2021, vol. 21, no. 18, p. 6125.

[34] Mohri, M., Rostamizadeh, A. and Talwalkar, A. *Foundations of machine learning*. Cambridge, MA: MIT press, 2018.

[35] Mok, M. S., Sohn, S. Y. and Ju, Y. H. Random effects logistic regression model for anomaly detection. *Expert systems with applications*. Amsterdam, Netherlands: Elsevier. 2010, vol. 37, no. 10, p. 7162–7166.

[36] Na, G. S., Kim, D. and Yu, H. DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams. In:. New York, NY, USA: Association for Computing Machinery, 2018, p. 1993–2002. KDD '18.

[37] Nasteski, V. An overview of the supervised machine learning methods. *Horizons. b*. 2017, vol. 4, p. 51–62.

[38] Nguyen, T. T., Nguyen, U. Q. et al. An evaluation method for unsupervised anomaly detection algorithms. *Journal of Computer Science and Cybernetics*. Hanoi, Vietnam: Vietnam Academy of Science and Technology. 2016, vol. 32, no. 3, p. 259–272.

[39] Pang, G., Shen, C., Cao, L. and Hengel, A. V. D. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*. NY, USA: ACM New York. 2021, vol. 54, no. 2, p. 1–38.

[40] Patel, V. R. and Mehta, R. G. Impact of outlier removal and normalization approach in modified k-means clustering algorithm. *International Journal of Computer Science Issues (IJCSI)*. Citeseer. 2011, vol. 8, no. 5, p. 331.

[41] Paulauskas, N. and Bagdonas, Ą. F. Local outlier factor use for the network flow anomaly detection. *Security and Communication Networks*. Wiley Online Library. 2015, vol. 8, no. 18, p. 4203–4212.

[42] Pokrajac, D., Lazarevic, A. and Latecki, L. J. Incremental Local Outlier Detection for Data Streams. In: *2007 IEEE Symposium on Computational Intelligence and Data Mining*. New York, NY, USA: IEEE, 2007, p. 504–515.

[43] Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W. et al. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*. IEEE. 2021, vol. 109, no. 5, p. 756–795.

[44] Salehi, M., Leckie, C., Bezdek, J., Vaithianathan, T. and Zhang, X. Fast Memory Efficient Local Outlier Detection in Data Streams. *IEEE Transactions on Knowledge and Data Engineering*. Washington, DC, USA: IEEE Computer Society. december 2016, vol. 28, p. 1–1.

[45] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*. Armonk, NY: IBM. 1959, vol. 3, no. 3, p. 210–229.

[46] Sander, J., Ester, M., Kriegel, H.-P. and Xu, X. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*. New York, NY, USA: Springer. 1998, vol. 2, p. 169–194.

[47] Schafer, R. W. What Is a Savitzky-Golay Filter? [Lecture Notes]. *IEEE Signal Processing Magazine*. IEEE. 2011, vol. 28, no. 4, p. 111–117.

[48] Schubert, E. and Gertz, M. Improving the Cluster Structure Extracted from OPTICS Plots. In: *LWDA*. 2018, p. 318–329.

[49] Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. d. et al. Data stream clustering: A survey. New York, NY, USA: Association for Computing Machinery. jul 2013, vol. 46, no. 1.

[50] Tuor, A. R., Baerwolf, R., Knowles, N., Hutchinson, B., Nichols, N. et al. Recurrent neural network language models for open vocabulary event-level cyber anomaly detection. In: *Workshops at the thirty-second AAAI conference on artificial intelligence*. 2018.

[51] Weber, R., Schek, H.-J. and Blott, S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: *VLDB*. 1998, vol. 98, p. 194–205.

[52] Xu, W., Huang, L., Fox, A., Patterson, D. and Jordan, M. I. Detecting large-scale system problems by mining console logs. In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132.

[53] Xu, Z., Kakde, D. and Chaudhuri, A. Automatic hyperparameter tuning method for local outlier factor, with applications to anomaly detection. In: IEEE. *2019 IEEE International Conference on Big Data (Big Data)*. 2019, p. 4201–4207.

[54] Zeufack, V., Kim, D., Seo, D. and Lee, A. An unsupervised anomaly detection framework for detecting anomalies in real time through network system's log files analysis. *High-Confidence Computing*. Amsterdam, Netherlands: Elsevier. 2021, vol. 1, no. 2, p. 100030.

[55] Zhao, Y., Nasrullah, Z. and Li, Z. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research*. JMLR, Inc. and Microtome Publishing. 2019, vol. 20, no. 96, p. 1–7.

[56] Zhu, J., He, S., He, P., Liu, J. and Lyu, M. R. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct 2023, p. 355–366.

[57] Šiklóši, M. *System Log Analysis for Anomaly Detection Using Machine Learn- ing*. Brno, CZ, 2020. Master's thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications.