CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Security Log Analysis with Explainable Machine Learning

Master's thesis in Computer Science and Engineering

Linus Aronsson
Aron Bengtsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Security Log Analysis with Explainable Machine Learning

Linus Aronsson
Aron Bengtsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Security Log Analysis with Explainable Machine Learning
Linus Aronsson
Aron Bengtsson

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Advisor: Rikard Bodforss, Bodforss Consulting AB
Examiner: Tomas Olovsson, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A word cloud in the shape of a lock of the most used words in this thesis.

Typeset in LaTeX
Gothenburg, Sweden 2021

Security Log Analysis with Explainable Machine Learning

Linus Aronsson
Aron Bengtsson

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Physical access control systems are implemented to restrict access in order to prevent attacks from happening in the physical space. These systems usually produce access logs that contain information to track accesses made by users. The access logs can however end up becoming large and difficult to interpret, making security assessment impractical for administrators and as a consequence, the logs are rarely inspected. The current method of detecting anomalies by manual inspection is often not a feasible approach in preventing attacks. For this reason, anomaly detection using machine learning is a method that can aid administrators in detecting attacks and being able to proactively prevent them from happening again. In this thesis, we first analyze users from a dataset of physical access logs and cluster them into groups with similar behavior based on their access pattern. Next, we train two LSTM autoencoder models for each cluster in order to detect anomalies of two different access sequence lengths. Finally, we evaluate the model with the help of a security expert from the industry by reviewing explanations produced using SHAP values. The results in this thesis show that our method was able to reduce the number of log events that need to be manually inspected by 95.6% in the given dataset. The results also show that the explanations provided by SHAP values was able to help in understanding what caused an anomaly. In conclusion, our proposed method is advantageous compared to manual inspection as it greatly reduces the amount of work required to detect anomalies, and the SHAP values are able to help security administrators to work in a more proactive manner.

Keywords: security, physical access control, anomaly detection, machine learning, deep learning, LSTM autoencoder, explainability, SHAP.

Acknowledgements

We would like to acknowledge everyone who has helped and supported us during this project. We would like to thank our advisor Rikard Bodforss at Bodforss Consulting AB for allowing us to work on this project and for his assistance throughout the project. His expertise in the security domain was very useful in discussions on how to evaluate the project. We would also like to thank our supervisor Magnus Almgren at Chalmers for his thorough feedback on writing our thesis and for his guidance during discussions on the project.

Linus Aronsson, Aron Bengtsson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

List of Listings

1
Introduction

This chapter introduces the thesis with some background on physical access control and anomaly detection. The motivation behind the project is also explained. Additionally, the problem description and the goals of the project is described. Finally, the limitations of the thesis are detailed and the structure of the thesis is outlined.

1.1 Background and Motivation

According to Verizon's data breach investigations report of 2020, 30% of breaches involves internal actors, 8% were caused by authorized users intentionally misusing privilege and 4% were caused by physical actions [1]. A survey from 2018 showed that 53% of organizations encountered insider attacks in the previous 12 months and 27% said insider attacks have become more frequent [2]. This type of threat often consist of espionage, sabotage and theft caused by malicious insiders having access to critical parts of an organization's assets. It is therefore important to have access control enforced, especially physical access control, in order to protect the organization's assets.

Many security systems implemented for physical access control produce log entries that contain data to track door accesses to rooms and facilities. While a single log entry can be small, a collection of them often become large and difficult to interpret, thus defeating the purpose of logging and making security assessment ineffective. Arguably for that reason, these log entries are rarely examined other than when an incident has already occurred. From a security standpoint, it is important for businesses and organizations to work in both a reactive and a proactive manner. For the purpose of working more proactively, there is a need for anomaly detection in physical access control. With the use of anomaly detection, it would be possible to alert the system of potential breaches as they occur instead of finding out later by manually inspecting the logs.

With the use of machine learning, anomaly detection has become possible when managing large amounts of data. When specifically using unlabeled data, anomaly detection is accomplished in an unsupervised manner. *Autoencoders* are therefore relevant for this purpose [3]. While machine learning techniques may be very effective at detecting anomalies, the output can be hard to explain. A framework to solve this problem exists called *SHapley Additive exPlanations* [4]. This framework can be applied to autoencoders [5].

1.2 Problem Description

Physical security measures are used to ensure the restriction of access to a physical space such as a property, building, room or other physical asset to authorized users [6]. Systems used by businesses and organizations are typically implemented with access card readers. These types of systems function in that each user possesses a personal access card with corresponding permissions for that user. When using an access card at a card reader, the permissions of the card is compared to the access level of the physical space. If the card matches the access level, the user has permission to enter and gains access to the physical space. Every action performed at a card reader is stored as a log entry. If a user accesses a physical space, information about the user together with a timestamp and information about the location is stored. In a perfect world, this would work without flaws, but this is unfortunately not the case. The system could be misconfigured and an access card could potentially be stolen and misused. Considering that these security logs are rarely inspected, but also that anomalies can be hard to discover by manual inspection, security breaches can easily develop and may even go unnoticed. A short summary of the different challenges in this project is listed below.

- **Access log analysis**. The logs need to be analyzed properly in order to provide an understanding of the different events, doors and user behavior in the data.

- **Data-driven model**. The model is purely data-driven as there is no additional information about the physical layout or user behavior beyond the used dataset. The idea is that a data-driven system is easy to build without requiring expensive experts to configure the system. The limitations of this approach will be further explored in this thesis.

- **Unlabeled dataset**. The dataset that is used in this project is unlabeled. As a result, there is no specific information that reveals if a data point is actually normal or anomalous. This is often the case in real-world settings since labeling is costly. The aim is therefore to consider a method that can be used by industry.

- **Model explainability**. Not only should the model detect anomalies, it should also provide an explanation of what possibly caused the anomalies. This should aid an administrator in preventing the anomaly from happening again.

1.3 Goals and Scope

This project primarily aims to create a working prototype of anomaly detection for physical access control logs using machine learning. An important aspect of this project is that the prototype should be explainable. This means that the model should not only output that an anomaly has been detected, but also output what sort of anomaly it is. The goal with this prototype is to improve existing

physical access control systems to have a richer analysis model other than manual inspection of access logs. The aim with this thesis is to answer the following questions:

1. What type of anomalies in the context of physical access control can a machine learning model detect?

2. How can explainability be achieved in a deep learning model?

3. How can explainable machine learning be used to proactively prevent security breaches?

1.4 Delimitations

Although it would be possible to develop a model that is generally applicable, as in being usable for anomaly detection in domains other than physical access control, this is not the focus of the project. This project will not involve any collection of data, as this will be provided by Bodforss Consulting AB that this project is done in collaboration with.

1.5 Thesis Outline

Chapter 1 provided a brief introduction to the topic of the thesis, along with the goals, scope and delimitations of the project.

Chapter 2 presents previous work and research related to this thesis. Furthermore, this chapter describes how this thesis is different from the previous research.

Chapter 3 introduces the different concepts and techniques that are used in this thesis. This chapter is intended to help the reader understand what is presented in subsequent chapters.

Chapter 4 presents the analysis performed on the physical access control logs. The main pieces of information from the logs are summarized in figures and tables. Finally, some conclusions are drawn about the main findings from the analysis.

Chapter 5 describes the architecture and development of the anomaly detection framework. This chapter also presents how the explainability aspect of the machine learning models were achieved. The chapter concludes with a description of how the framework is evaluated.

Chapter 6 presents the results of the thesis.

Chapter 7 discusses the results of the thesis and concludes with insight on the topic and future work.

2
Related Work

Anomaly detection in data is a research area that has been studied as early as the 19^{th} century in statistics [7]. Denning [8] proposed anomaly detection in security for intrusion detection systems in 1987. In more recent years, anomaly detection has been researched with regard to the application of machine learning. Sommer and Paxson [9] discusses the application of machine learning for network intrusion detection in terms of anomaly detection. In the paper they point out that, despite extensive research on applying machine learning to anomaly detection, the deployment of such systems are limited in operational settings. They mention one reason for the limited deployment being that there is a clear difference in the objective of traditional machine learning tasks compared to anomaly detection. Machine learning is traditionally about finding similarities between data samples rather than dissimilarities, which is the case in anomaly detection. Particularly, attempting to find dissimilarities involves finding outliers, which may be a rare occurrence in the data. Achieving acceptable results are thus significantly harder considering that a common principle in machine learning is that the number of data samples of each class to be identified must generally be large in order to achieve anything of interest. Another reason is that evaluation of anomaly detection systems are crucial but difficult to perform. The authors refer to the evaluation as being more difficult than developing the system itself. The lack of publicly available labeled datasets for research purposes regarding anomaly detection further contributes to the difficulty.

There are several papers involving anomaly detection specifically for log file analysis. Frei and Rennhard [10] proposed a system that uses a combination of graphical and statistical techniques to visualize the content of a log file in order to aid security administrators in identifying anomalies. Juvonen *et al.* [11] proposed a framework using dimensionality reduction techniques, as log files may consist of high-dimensional data, in order to find anomalies in HTTP logs. There are different techniques applied to detecting anomalies in the context of physical access. Davis *et al.* [12] used a graph model for anomaly detection and applied it to physical access logs from an office building. They present an algorithm that searches labeled graphs for both structural anomalies, described as unusual paths through the building, and numerical anomalies, described as unusual timing data. Cheh *et al.* [13] used a Markov model to detect malicious insiders using physical access logs in a railway station together with knowledge about the physical layout of the station. Similar to our model, users are distinguished based on their past behavior. The reason being that access control systems assign roles to users by the doors they have permission to access, which does not necessarily reflect the user behavior. Poh *et al.* [14] used a

clustering model to identify anomalous user behavior by characterizing users based on physical movement from access logs in an office building together with their job profile.

Our model is different from these papers in that we do not have explicit knowledge about the physical layout nor do we have accurately defined job profiles apart from access roles. Additionally, we apply *Deep Learning* [15] in the form of an artificial neural network called an autoencoder to detect anomalies. We also use SHapley Additive exPlanations to achieve explainability for the output. This is important for our model since someone who is not necessarily an expert in machine learning should be able to understand why an event was considered an anomaly. Another important aspect of our model is that it is purely data-driven, meaning that it can be applied to many systems that lacks further information about the physical layout and user behavior aside from what can be found in the access logs.

3
Theory

This chapter presents the theory behind the different concepts and techniques used in this project. Section 3.1 introduces the concept of access control, including the different security threats in a physical environment. Section 3.2 then describes the concept of anomaly detection and defines the different types of anomalies. This project primarily uses different machine learning techniques. Section 3.3 therefore introduces the topic of machine learning and describes some of the relevant concepts within the subject. Finally, Section 3.4 presents the specific machine learning models that are utilized.

3.1 Access Control

Access control is the security process of restricting access to a specific resource, where requests to access the given resource can be granted or denied [16]. The restriction is normally policy-driven and accessing resources is resolved using identity. The policies in access control systems are usually user-specific, meaning that authentication is performed by users. Authorization is achieved in a number of different ways depending on the access control model. The model that is of interest in this project is role-based access control, especially for physical access.

3.1.1 Role-Based Access Control

In role-based access control (RBAC), permission to access resources are based on a user's role, which is usually established from the responsibilities that the user has within an organization [17]. This model is beneficial in business as roles are easy to organize based on the business structure and the work duties of the employees. RBAC also aligns with the principle of least privilege since each role can be assigned the minimum level of permission required for the user to carry out their duty. A problem with RBAC is that the roles must be maintained and documented. If the responsibilities of a user changes, the role of the user needs to be updated in order to ensure that the user does not have more privilege than necessary.

3.1.2 Physical Access Control

Physical access control (PAC) refers to restricting access to a physical space. On a basic level, this can be seen as requiring a key to open a door with a lock. PAC systems that are controlled by software using electronic locks and access cards to

grant access can be considered as logical access control [18]. PAC systems are thus able to be controlled centrally, which allows permissions of a user's physical access to be modified or revoked. Users are granted access to a physical space by using a valid access card at an electronic lock equipped with a card reader. The credentials of the card, such as an identification number or an access role, are checked against a database to see if the user has permission to enter.

3.1.3 Physical Security Threats

There are different types of physical security threats, such as environmental threats and human-caused threats. In this project, only the physical threats caused by humans are relevant. Human-caused threats can be hard to predict and the potential damage of an attack can be severe. This type of threat can be grouped into a few categories [19]:

- **Unauthorized physical access.** Any individual that has entered a physical space without permission is to be considered a threat that can potentially lead to theft, vandalism or misuse. One way for an unauthorized individual to access a restricted area is by following an authorized user, known as tail-gating [20]. This can be very hard to detect unless the area is equipped with video surveillance, since this access does not appear in security logs from the PAC system.

- **Theft.** This threat includes any theft of equipment and theft of data. This can be perpetrated by an authorized insider or by an unauthorized outsider. Theft also includes eavesdropping, where a user obtains sensitive information without permission.

- **Vandalism.** This threat involves any destruction of property, destruction of equipment and destruction of data.

- **Misuse.** This threat includes incorrect use of resources, such as performing a malicious action against a system. This can be performed by users who are authorized to use the resources, as well as individuals who are not authorized to use the resources at all.

3.2 Anomaly Detection

Anomaly detection is the task of identifying patterns in data that is different from the expected behavior [21]. These patterns are normally referred to as anomalies or outliers. The anomaly usually corresponds to some kind of problem. In physical access control, this could could correspond to one of the threats mentioned in Section 3.1.3. Many different anomaly detection techniques have been proposed across several research communities. Some of these techniques are domain specific, while others work in a more general sense. The details on how anomaly detection was performed in this project is specifically described in Chapter 5.

3.2.1 Types of Anomalies

Anomalies can be arranged into three different categories: *point anomalies*, *collective anomalies* and *contextual anomalies* [21].

Point Anomalies

An event is considered a point anomaly if the event is anomalous with respect to other events in the data. The event is therefore an anomaly since it is different from normal events. For example, in physical access control, a door where the number of accesses for a specific user is unusually high compared to the number of accesses of peers is considered a point anomaly and could potentially be a threat.

Collective Anomalies

A collection of related events is considered a collective anomaly if the collection is anomalous with respect to the rest of the data. One event in a collective anomaly might not be an anomaly by itself, but its occurrence in conjunction with other events is anomalous. For example, one door access might not be an anomaly on its own, but a sequence of door accesses could be anomalous.

Contextual Anomalies

An event is considered a contextual anomaly if the event is anomalous in a specific context but not otherwise. The definition of a context depends on the data and has to be specified before being able to detect contextual anomalies. The data can be described using two sets of attributes:

- **Contextual attributes.** These attributes specify the context of the events in the data. For example, this could be the physical location of an event in spatial data or the time in time-series data.

- **Behavioral attributes.** These attributes specify the non-contextual features of the events in the data. For example, this could be the weather at a location in spatial data or the temperature at a given time in time-series data.

An anomaly is determined using the values of the behavioral attributes within a specific contextual attribute. An event might be considered an anomaly in one context, but could be normal in a different context. For example, a door access during the day might be normal, but during the night this would be considered an anomaly. Important to note is that a point anomaly or a collective anomaly can also be deemed a contextual anomaly if context is taken into consideration.

3.3 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) and refers to the study of algorithms that are able to learn from data and improve their performance through experience of a given task. To understand what learning means in this context, a definition is provided by Mitchell [22]:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

The tasks in T refers to what the algorithm is intended to accomplish and are usually described in terms of how the algorithm should process an *example* [15]. An example is a set of *features* measured from an object or event. Typically, an example is represented as a vector $\boldsymbol{x} \in \mathbb{R}^n$ called a *feature vector* that consists of a number of numerical features. The performance measure P is used to evaluate how effective the algorithm is at learning a specific task T. The *accuracy* of the machine learning model is often used as a performance measure. Accuracy is measured as the percentage of examples where the model produces the correct output. Another similar performance measure that is often used is the *error rate* of the model, which is the percentage of examples where the model produces the incorrect output. The experience E is the data that is available during the learning process of the algorithm. Machine learning algorithms can generally be categorized as supervised learning or unsupervised learning, depending on the allowed experience. The algorithms are usually allowed to experience a complete *dataset* consisting of many examples.

3.3.1 Supervised Learning

In supervised learning, algorithms receive datasets where each example is a pair of an input object ($\boldsymbol{x}$) and an output value ($\boldsymbol{y}$) called a *label* or *target* [15]. The structure of a dataset with n pairs is shown in (3.1).

$$\{(\boldsymbol{x}_1,\ \boldsymbol{y}_1),\ (\boldsymbol{x}_2,\ \boldsymbol{y}_2),\ \ldots\ ,\ (\boldsymbol{x}_n,\ \boldsymbol{y}_n)\} \in (\boldsymbol{X} \times \boldsymbol{Y})^n \tag{3.1}$$

With $\boldsymbol{X}$ being the input space and $\boldsymbol{Y}$ being the output space, the goal of supervised learning is to infer a function $f : \boldsymbol{X} \to \boldsymbol{Y}$ such that each input $\boldsymbol{x_i}$ is mapped to the correct output $\boldsymbol{y_i}$. The function f is then applied to predict the output for unseen data [23]. Supervised learning can be divided into two types of problems, *regression* and *classification*.

Regression

For regression problems, the learning algorithm infers a function $f : \mathbb{R}^n \to \mathbb{R}$ to predict numerical values based on the input.

Classification

For classification problems, the learning algorithm infers a function $f : \mathbb{R}^n \rightarrow \{1, \ldots, k\}$ to determine which one of k discrete classes the input belongs to. A deterministic approach to classification only outputs the most likely discrete class, whereas a probabilistic approach outputs a probability distribution over the classes. The probabilistic approach is usually done by estimating the posterior probability $\boldsymbol{P(y \mid x)}$ of each class $\boldsymbol{y} \in \boldsymbol{Y}$ given an input $\boldsymbol{x} \in \boldsymbol{X}$ [15]. Some probabilistic classifiers use a joint probability distribution model $\boldsymbol{P(x, y)}$ to estimate the posterior probability. Other probabilistic classifiers use a conditional probability model $\boldsymbol{P(y \mid x)}$ to determine the posterior probability directly. Classifiers based on joint probability distribution are called *generative* and classifiers based on conditional probability are called *discriminative* [24]. A probabilistic classifier can seem deterministic by returning the class $\boldsymbol{y}$ with the highest probability score.

3.3.2 Unsupervised Learning

In unsupervised learning, algorithms receive datasets where each example is simply an input object ($\boldsymbol{x}$) without an associated target output. The goal of unsupervised learning is to find patterns in the data outside of what might otherwise be considered unstructured noise. Two essential parts of unsupervised learning are *cluster analysis* and *dimensionality reduction* [25].

Cluster Analysis

Cluster analysis, also known as clustering, is the task of grouping a set of examples into groups called clusters, where the examples within a cluster are more similar to each other than to examples in other clusters. The similarity between examples is measured using various distortion or distance measures, depending on the clustering algorithm used [26]. There is no precise definition of a cluster, and as such, the notion of a cluster is, largely, in the eye of the beholder [27]. As a consequence, the notion of a cluster depends on the clustering algorithm and its properties. There are many different clustering algorithms which can be categorized based on their type of model. The clustering method used in this project is *agglomerative hierarchical clustering* (see Section 3.4.1).

Dimensionality Reduction

Dimensionality reduction is the task of mapping an n-dimensional point into a lower k-dimensional space, where the n-dimensional point is a representation of an example with n features in an n-dimensional space [28]. The goal of dimensionality reduction is to generate the k dimensions while retaining the important properties of the original example. When performing dimensionality reduction on an entire dataset, the relationship between the examples in the original n-dimensional space must also be preserved. The reason behind using this technique is because working with high-dimensional is usually intractable. Dimensionality reduction can be useful

in a number of different cases, such as cluster analysis, data visualization, especially when mapping data onto two or three dimensions, and removing noise from data. Dimensionality reduction is usually divided into *feature selection* and *feature extraction* depending on the approach [29], with models being either linear or nonlinear [30].

- **Feature Selection**. This approach is based on only retaining the most important features, meaning that redundant and irrelevant features are omitted.

- **Feature Extraction**. This approach considers all features and transforms the data into a lower dimensional space where the dataset structure is retained as best as possible.

In this project however, only feature extraction will be utilized. An example of a linear feature extraction model is *principal component analysis* (see Section 3.4.2) and an example of a nonlinear feature extraction model is *t-distributed stochastic neighbor embedding* (see Section 3.4.3).

3.3.3 Overfitting and Underfitting

A machine learning model is said to *overfit* if the model performs well on the training data but does not generalize well to new data, meaning that the model learns noisy or irrelevant features rather than the underlying structure of the data [31]. The aim of a model is being able to perform well on test data that was not experienced during training. However, due to overfitting, which leads to a loss of accuracy on unseen data, the model will not be able to perform well on the test data. If the training data is noisy or too small, the model is likely to detect patterns in the noise which might not exist in new data. Overfitting generally occurs when the model is too complex relative to the noise in the training data. There are some approaches to counteract overfitting, such as simplifying the model, using a larger dataset during training and reducing noise in the training data by removing outliers. Simplifying a model by constraining it is called *regularization* and various machine learning models use different techniques to achieve this.

Underfitting is the opposite of overfitting: instead of the model being too complex, it is too simple to learn the underlying structure of the data [31]. In the case of underfitting, the model will not be able to perform well on the test data nor the training data. Underfitting can be counteracted by selecting a more complex model, reducing the regularization of the model and using more appropriate features for the algorithm.

3.3.4 Dataset Shift

Dataset shift is a problem that appears when the joint distribution differs between the training data and the test data [32], such that $P_{train}(y, x) \neq P_{test}(y, x)$. A common assumption is that the training data and the test data follow the same distribution. This is not always the case however, especially in real-world settings. As a result, this

problem leads to a decline in model performance. For example, in physical access control, this problem could appear if some users disappear over time or if an overall change of behavior occurs. The model would then train on data that is no longer similar in the test data. Dataset shift can be categorized into different types. Three of these types will be described as they are the most commonly present in real-world settings: *covariate shift*, *prior probability shift* and *concept shift* [33].

Covariate Shift

Covariate shift occurs when the distribution of the input variables $P(x)$ changes between the training data and the test data. Formally, this can be defined as the case where:

- $P_{train}(y \mid x) = P_{test}(y \mid x)$ and $P_{train}(x) \neq P_{test}(x)$.

Prior Probability Shift

Prior probability shift can be thought of as the opposite of covariate shift. This type of dataset shift occurs when the distribution of the target variables $P(y)$ changes between the training data and the test data. Formally, this can be defined as the case where:

- $P_{train}(x \mid y) = P_{test}(x \mid y)$ and $P_{train}(y) \neq P_{test}(y)$.

Concept Shift

Concept shift, also referred to as concept drift, occurs when the relationship between the input and the target variables changes over time. Formally, this can be defined as two different cases:

- $P_{train}(y \mid x) \neq P_{test}(y \mid x)$ and $P_{train}(x) = P_{test}(x)$.
- $P_{train}(x \mid y) \neq P_{test}(x \mid y)$ and $P_{train}(y) = P_{test}(y)$.

3.3.5 Feature Engineering

Feature engineering is the process of using domain knowledge to define the most appropriate features given the data, the task and the model used [34]. Generally, this is an iterative process that includes the following steps:

1. *Brainstorm* what features to use.
2. *Create* the features.
3. *Evaluate* how the features work with the model.
4. *Repeat* the process if necessary.

Creating relevant features is important in order for the model to produce desired result and to avoid overfitting and underfitting. Feature engineering typically consumes a

lot of time, but if carried out adequately, other parts of the machine learning process may become easier. There are techniques that uses *Deep Feature Synthesis* [35] to automate feature engineering, but this will not be explained in further detail as this has not been utilized in this project.

3.3.6 Data Preprocessing

In order to analyze data and use it as input to machine learning models, the data must first be prepared and organized into a suitable form. This process is known as *data preprocessing* and is a critical part of machine learning as results would otherwise be misleading or incorrect [36]. The type of data preprocessing varies considerably depending on the raw data and the specific models that are applied. This section will therefore be limited to describing *one-hot encoding* and *data normalization*.

One-Hot Encoding

One-hot encoding is a technique used to deal with categorical data [37]. Transforming data from categorical to numerical is an important step in preprocessing since machine learning models often require their input to be numerical values. Consider a categorical variable $\boldsymbol{x}$ with n distinct values $x_1, x_2, \ldots, x_n$. The one-hot encoding of a specific value x_i is a vector $\boldsymbol{v}$ where each element has a value of 0 except for the ith element that has a value of 1. For example, assume that the categorical variable $\boldsymbol{x}$ is *color* with possible values from the set S = {red, green, blue}. If x_1 = red, x_2 = green and x_3 = blue, then the one-hot encoding of $\boldsymbol{x}$ is (1, 0, 0), (0, 1, 0) and (0, 0, 1) (see Figure 3.1).

id	color
1	red
2	green
3	blue

⇨

id	red	green	blue
1	1	0	0
2	0	1	0
3	0	0	1

Figure 3.1: Transformation from original data to one-hot encoded data.

Data Normalization

Raw numerical data is often not in a suitable form to be processed properly by machine learning techniques as the range of values varies between different features. Data normalization is used to transform raw numerical data to a suitable form while retaining differences in the values of each feature [36]. Normalization is an important part of preprocessing since features with different range of values otherwise will influence the result of machine learning algorithms disproportionately. A common method to achieve normalization in the interval $[0, 1]$ or $[-1, 1]$ is *min-max normalization*. This method projects the original range of values onto the new range. The min-max normalization in the interval $[0, 1]$ is given by (3.2), where $\boldsymbol{x}$ is the original

feature and $\boldsymbol{x}_{norm}$ is the normalized feature.

$$\boldsymbol{x}_{norm} = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \tag{3.2}$$

3.3.7 Explainability and Interpretability

Explainability and interpretability are often used interchangeably in literature [38], arguably because a collective understanding of explainability as a concept still needs to develop [39]. Some papers however, point out the importance of making a distinction between the two terms [40]. This thesis uses the definition of an explanation from [41]:

> *An explanation is the collection of features of the interpretable domain, that have contributed for a given example to produce a decision (e.g. classification or regression).*

As such, explainability can be defined as providing an understanding of what features contributed to the output of a machine learning model. An interpretation on the other hand can be defined as [41]:

> *An interpretation is the mapping of an abstract concept (e.g. a predicted class) into a domain that the human can make sense of.*

Interpretability can therefore be defined as presenting the properties of a machine learning model in a way that is understandable to humans. Both explainability and interpretability are essential concepts in order to achieve *explainable* machine learning. It is important to understand what features contributed to the output, and at the same time, this needs to be presented in a way that a human can comprehend it. One framework that achieves explainability is SHapley Additive exPlanations, which is used in this project.

SHapley Additive exPlanations

SHapley Additive exPlanations (SHAP) is a model-agnostic framework for explaining the output of machine learning models [4]. Being model-agnostic means that the framework is not concerned by what goes on inside of the model, therefore being agnostic to the internals of the model. Explaining the output is achieved by computing the contribution of each feature to the output, thus assigning each feature an importance value. SHAP is based on the classic *Shapley values* from game theory and represents the Shapley value explanation as a linear model called an *additive feature attribution method*. This explanation is given by (3.3), where g is the explanation model, $z' \in \{0,1\}^M$ is the simplified input vector, M is the number of simplified input features and $\phi_i \in \mathbb{R}$ is the feature attribution for a specific feature i, the

Shapley values.

$$g\left(z'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i' \tag{3.3}$$

The goal of the explanation model is to explain the output $f(x)$ given an input x to the original model f. Explanation models generally use simplified inputs x' that are mapped to the original input x using a mapping function $x = h_x(x')$. In the input vector, a value of 1 corresponds to a present feature and a value of 0 corresponds to a missing feature. For x, the input vector x' corresponds to all present features. The explanation for x can then be simplified as (3.4).

$$g\left(x'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{3.4}$$

The unique solution that defines the Shapley values is given by (3.5), where $|z'|$ is the number of present features in z' and $z' \subseteq x'$ represents all z' vectors where the present features are a subset of the present features in x'.

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left[f_x\left(z'\right) - f_x\left(z' \setminus i\right)\right] \tag{3.5}$$

SHAP solves this equation in order to compute its values, where $f_x(z') = f(h_x(z')) = E\left[f(z) \mid z_S\right]$. Here, S is the set of present features in z'. A simplified input mapping function $h_x(z') = z_S$ is used, where z_S has missing values for features not in S. $E\left[f(z) \mid z_S\right]$ is an approximation of $f(z_S)$ since most models are not able to manage arbitrary patterns of missing input values. SHAP values explain how to get from a base value $E\left[f(z)\right]$ to the original output value $f(x)$ (see Figure 3.2). The base value is the output when no features are known.

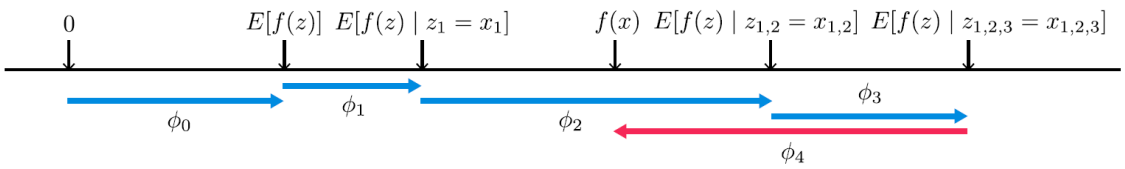


Figure 3.2: SHAP values illustrating the contribution of each feature to the output. From [4].

3.3.8 Loss Functions

A loss function, also known as a cost function, is a function that measures the performance of a machine learning model for given data by determining the *loss* [42]. The loss is an error value between a predicted output value $\hat{\boldsymbol{y}}$ and the correct output value $\boldsymbol{y}$. In this project, a loss function called Smooth L1 loss is used. In order to

put this loss function into context, L1 loss will first be described.

L1 Loss

L1 loss is a loss function based on minimizing *mean absolute error* (MAE) [43]. MAE computes the sum of absolute differences between the predicted output values and the correct output values. In other words, it measures the average magnitude of errors across the predicted output values. MAE is given by (3.6), where n is the number of data points in the dataset, $\hat{y}_i$ is the predicted output value for instance i and y_i is the correct output value.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.6}$$

L1 loss is robust to outliers, but it suffers from not being differentiable. Specifically, the derivative of the absolute value $f(x) = |x|$ is not defined at each point in its domain (see Equation 3.7). As can be seen, the derivative is undefined for $x = 0$.

$$\frac{d}{dx}|x| = \begin{cases} -1, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{3.7}$$

For that reason, the problem with L1 loss is that its gradient is constant. As such, even when the loss value is small, the gradient will be large, causing the loss function to be unstable. This makes it difficult to converge effectively.

Smooth L1 Loss

Smooth L1 loss, introduced by [44], is a loss function with the same advantage as L1 loss of being robust to outliers. Smooth L1 loss is given by (3.8), where x is the difference between the predicted output value and the correct output value.

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{3.8}$$

The difference between the original L1 loss and Smooth L1 loss is that the latter is stable due to being differentiable (see Equation 3.9).

$$\frac{d}{dx}\,\text{smooth}_{L_1}(x) = \begin{cases} x, & \text{if } |x| < 1 \\ \pm 1, & \text{otherwise} \end{cases} \tag{3.9}$$

3.4 Machine Learning Models

This section describes the different models that are used in this project. First, a clustering method called agglomerative hierarchical clustering is described. Next, two dimensionality reduction techniques called principal component analysis and t-distributed stochastic neighbor embedding are detailed. Subsequently, a type of

artificial neural network called a recurrent neural network is specified. Finally, a neural network architecture called an autoencoder is described.

3.4.1 Agglomerative Hierarchical Clustering

The agglomerative hierarchical clustering algorithm constructs clusters by recursively partitioning data points in a bottom-up approach [45]. Each data point starts as its own cluster. The algorithm then proceeds to successively merge the clusters into larger ones. Which clusters to merge is decided based on some similarity measure that is chosen to optimize some linkage criterion. In this project, *Ward's method* [46] was used. At each step, Ward's method minimizes the sum of squared differences within all clusters. The pair of clusters that yield the smallest increase in variance during the step will then be merged. This process continues until all data points end up in a single cluster or until a condition is fulfilled, such as reaching a set number of clusters. The hierarchy of the clusters can be visualized as a dendrogram (see Figure 3.3).

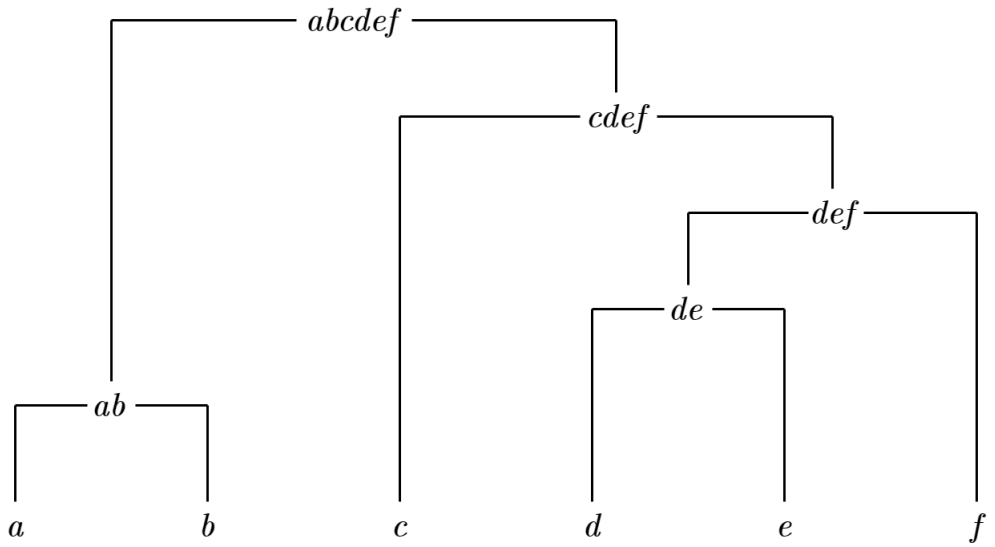


Figure 3.3: Dendrogram of a hierarchical clustering.

3.4.2 Principal Component Analysis

Principal component analysis (PCA) is used to reduce the dimensionality of a dataset, while retaining as much of the variation in the data as possible. This is achieved by transforming the data to a new set of variables, called the principal components (PCs), that are linear functions of those in the original dataset, that maximize variance and are uncorrelated [47]. The context for PCA involves a dataset with p numerical variables for each of n features, defining an $n \times p$ data matrix $\boldsymbol{X}$. The ith column of $\boldsymbol{X}$ is the vector $\boldsymbol{x}_i$ of observations on the ith variable. Given a vector $\boldsymbol{x}$, PCA starts by computing a linear function $\boldsymbol{\alpha}_1^T\boldsymbol{x}$ of the variables of $\boldsymbol{x}$ having maximum variance, where $\boldsymbol{\alpha}_1$ is a vector of p constants $\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1p}$ and T denotes transpose. The

linear function is given by (3.10).

$$\boldsymbol{\alpha}_1^T \boldsymbol{x} = \alpha_{11} x_1 + \alpha_{12} x_2 + \ldots + \alpha_{1p} x_p = \sum_{i=1}^{p} \alpha_{1i} x_i \tag{3.10}$$

PCA then computes $\boldsymbol{\alpha}_2^T \boldsymbol{x}$ having maximum variance, uncorrelated with $\boldsymbol{\alpha}_1^T \boldsymbol{x}$ and so on. At the kth computation, the kth PC $\boldsymbol{\alpha}_k^T \boldsymbol{x}$ having maximum variance is uncorrelated with the previous $\boldsymbol{\alpha}_1^T \boldsymbol{x}, \boldsymbol{\alpha}_2^T \boldsymbol{x}, \ldots, \boldsymbol{\alpha}_{k-1}^T \boldsymbol{x}$ PCs. It is possible to find p PCs, but the goal is generally to find m PCs, where $m \ll p$ and the variation in $\boldsymbol{x}$ is accounted for [48].

3.4.3 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique well-suited for visualizing high-dimensional data by mapping each datapoint to a location in a two or three-dimensional space [49]. The technique is based on Stochastic Neighbor Embedding (SNE) originally presented by Hinton and Roweis [50]. The loss function of t-SNE is a symmetrized version of the loss function used by SNE with simpler gradients. It also uses a t-distribution to compute the similarity between two points in the low-dimensional space, instead of a Gaussian distribution like SNE. The t-SNE algorithm starts by converting the similarities between datapoints to joint probabilities where similar datapoints are assigned a higher probability and dissimilar datapoints are assigned a lower probability. The algorithm then minimizes the Kullback-Leibler divergence between the joint probabilities of the low-dimensional space and the high-dimensional data.

3.4.4 Recurrent Neural Network

A *recurrent neural network* (RNN) is a type of artificial neural network that introduces a notion of time to the model [51]. Unlike traditional neural networks, RNNs are able to use information from previous states to influence the current output. This functions as a sort of memory as it can remember things learnt from prior input. RNNs can use their memory to process sequential data. This can be visualized by unfolding the network (see Figure 3.4), which can be trained using backpropagation. The specific algorithm used by RNNs is called *backpropagation through time* (BPTT) [52].

In the unfolded network, each individual layer represents a time step. At time step $\boldsymbol{t}$, the hidden node $\boldsymbol{h}_t$ receives input from the current data point $\boldsymbol{x}_t$ and from the hidden node $\boldsymbol{h}_{t-1}$ in the previous state. The output $\hat{\boldsymbol{y}}_t$ is then calculated using the hidden node value $\boldsymbol{h}_t$. The hidden node value $\boldsymbol{h}_t$ is given by (3.11) and the output value $\hat{\boldsymbol{y}}_t$ is given by (3.12). Here, $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are activation functions. The matrices W_{hx}, W_{hh} and $W_{\hat{y}h}$ are shared weight parameters between layers. The vectors $\boldsymbol{b}_h$ and $\boldsymbol{b}_{\hat{y}}$ are bias parameters that allow nodes to learn an offset.

$$\boldsymbol{h_t} = \boldsymbol{a}_1 \left(W_{hx} \boldsymbol{x}_t + W_{hh} \boldsymbol{h}_{t-1} + \boldsymbol{b}_h \right) \tag{3.11}$$

$$\hat{\boldsymbol{y}}_t = \boldsymbol{a}_2 \left(W_{\hat{y}h} \boldsymbol{h}_t + \boldsymbol{b}_{\hat{y}} \right) \tag{3.12}$$

A problem with RNNs is that they can encounter the *vanishing gradient problem* during training [53]. The reason why this occurs is because it is difficult to capture long-term dependencies. In short, when introducing many time steps, a small gradient will exponentially decrease. The weight parameters will eventually become insignificant (i.e. 0) when updated. At this point, the network is no longer learning. A solution to this problem was presented by Hochreiter and Schmidhuber called *Long Short-Term Memory* (LSTM) [54]. LSTM is the type of network that will be utilized in this project.

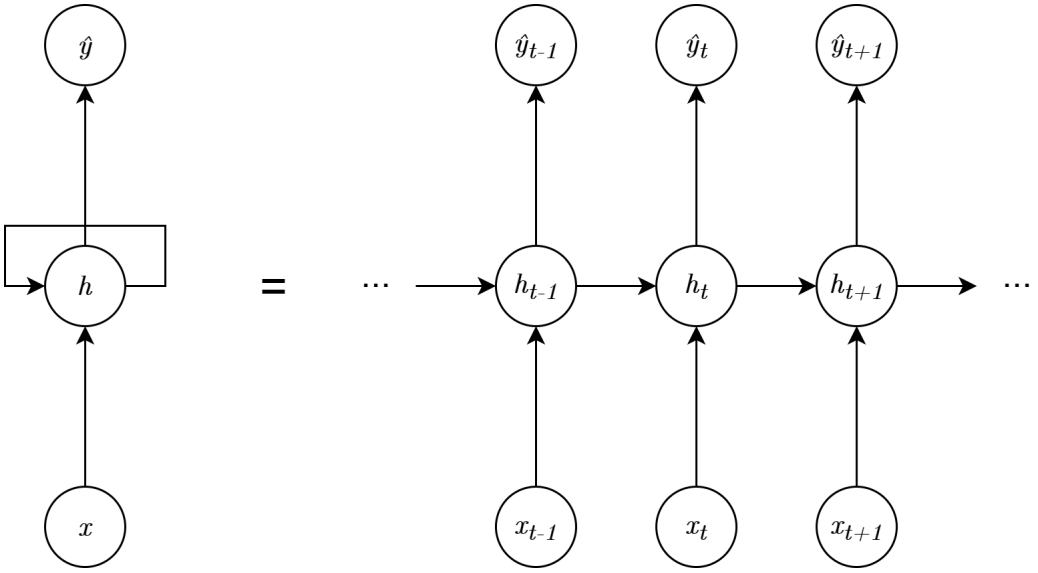


Figure 3.4: Basic RNN architecture. Left: Rolled RNN. Right: Unfolded RNN over time.

3.4.5 Autoencoder

An autoencoder is a neural network that consists of two parts: an *encoder* that maps an input to a hidden layer $\boldsymbol{h} = f(\boldsymbol{x})$ called the *code*, and a *decoder* that produces a reconstruction $\boldsymbol{r} = g(\boldsymbol{h})$ of the input from the hidden layer [15]. The two parts are trained jointly in an unsupervised manner with the aim of learning an approximate representation of the original input. The autoencoder is trained to minimize the reconstruction error $L(\boldsymbol{x}, \boldsymbol{r})$, which measures the dissimilarity between the input and the reconstruction.

LSTM Autoencoder

An LSTM autoencoder is a type of autoencoder that uses an encoder-decoder LSTM architecture to process sequential data. The encoder maps an input sequence to a fixed-length vector and the decoder maps the vector representation to a target sequence. This architecture was developed for natural language processing problems [55]. In this project however, the architecture will be applied to anomaly

detection.

Autoencoders for Anomaly Detection

If an autoencoder always succeeds in reconstructing the input without error, such that $g(f(\boldsymbol{x})) = \boldsymbol{x}$, simply by memorizing the input and passing it along the network, then it would not be particularly useful. Autoencoders are therefore usually restricted to reconstruct the input only approximately, essentially forcing the network to prioritize the most frequent properties of the data. This is what makes autoencoders relevant for anomaly detection. When encountering an anomaly, the autoencoder will not be able to reconstruct it accurately since outliers are not prioritized during the learning process. As a result, the reconstruction error will be low for normal data and high for anomalous data. By defining a threshold T, any data with a reconstruction error below the threshold $L < T$ can be considered normal, whereas any data with a reconstruction error above the threshold $L > T$ can be considered anomalous.

4
Analysis of Access Logs

As already mentioned, this project is purely data driven. In other words, no prior information, including the layout of the building or common user behavior, is available. All information will be retrieved based on analysis of the access logs. This chapter presents how this analysis was executed, and summarizes the main pieces of information in various plots and tables.

Section 4.1 begins by briefly describing what information will be presented, and motivates why this information is extracted in the context of this project. Section 4.2 describes the original data format of the physical access logs. Section 4.3 describes the different events that appear in the logs. Section 4.4 presents the different doors that exist in the system. Section 4.5 outlines how different user behavior can be extracted from the original access logs, and presents some analysis of such behavior. Finally, Section 4.6 summarizes the findings of the chapter.

4.1 Introduction

The collaborating company Bodforss Consulting AB has provided a dataset of physical access logs. These logs contain all recorded events from a physical access control system deployed in a company building. The available logs stretch from 2018-02-14 to 2020-08-26, which is a total of 924 days. Table 4.2 presents a few statistics regarding the logs, which are discussed in more detail throughout the chapter. This section describes the most relevant information in these access logs for the purposes of this project. The information extracted and motivations for their extraction is listed below.

- **Log events**. An understanding of the original format of the physical access logs must first be established. This allows for further extraction of the most relevant parts for this project. The original log format is briefly explained in Section 4.2. The access logs consist of a number of distinct events, which are presented in Section 4.3 in order to understand which of them are relevant for further analysis in this project.

- **Doors in the building**. A detailed understanding of which doors that exist in the building is important as all the other analysis steps are heavily dependent on this. Section 4.4 will therefore present the doors and their relative frequencies in the log data.

Anomalies	
Type	**Description**
Contextual	Door access at an unusual time
Point	First time accessing a door
Point	Denied door access
Point	Unusual number of door accesses
Collective	Unusual sequences of door accesses
Collective	Short time interval between accesses of different doors
Collective	Repeated door accesses

Table 4.1: Specific anomalies defined in this project.

No. Entries	No. Events	No. Users	Start	End	No. Days
1,073,284	50	671	2018-02-14	2020-08-26	924

Table 4.2: General statistics regarding the physical access logs.

- **Overall user behavior**. The purpose of the project is to detect unusual behavior in these access logs. What constitutes unusual is highly context dependent. After discussions with domain experts from Bodforss Consulting AB, there are a number of anomalies that they consider relevant in this particular case. These anomalies are summarized in Table 4.1. Before detecting unusual behavior, one must first establish what is normal. Section 4.5 therefore presents analysis of common user behavior in the access control system. Additionally, an approximate layout of the building is shown based on the analysis.

- **Categorization of user behavior**. Eventually, a number of machine learning models will be used to detect anomalies in the log data, as is described in Chapter 5. A natural approach to this is to apply one machine learning model per subgroup of users that tend to behave similarly (Chapter 5 will describe the reason for this in detail). In order to categorize the behavior of different users, three different user criteria are considered. These include which doors the user tend to open, which weekdays they open them on and what time during the day they open them. The reason for these criteria is that they are necessary in order to capture behavior that in turn reflect the anomalies listed in Table 4.1. Section 4.5.3 presents some initial analysis of how clusters of users can be found based on these criteria.

4.2 Log Format

All logs from the physical access control system are stored as XML files, where each file is separated by date consisting of events that occurred during that specific date. A file has a root node called `<events>` that consists of a number of `<batch>` elements where each batch element contains an `<event>` element with some `<argument>` elements. An event includes a couple of attributes, the interesting attributes being `name`

```
<batch>
    <event name="access.card.valid.standard" timestamp="1615649437000">
        <argument value="door_name" type="door" id="1"/>
        <argument value="user_name" type="person" id="1"/>
        <argument value="access_category_name" type="access_category"/>
    </event>
</batch>
```

Listing 1: Example of a door access event using an access card.

```
<batch>
    <event name="door.requestToExit" timestamp="1615649437000">
        <argument value="door_name" type="door" id="1"/>
    </event>
</batch>
```

Listing 2: Example of a door access event using an exit button.

and `timestamp`. The name attribute is a string that describes the type of event that has occurred and the timestamp attribute is the time of when the event occurred. An argument includes the attributes `type`, `value` and `id`. The type attribute is a string describing the type of the argument, the value attribute specifies a readable representation of the object related to the event and the id attribute is the primary key for the object.

Some of the interesting events related to door accesses include `door.requestToExit` that occurs whenever a user unlocks a door using an exit button and `access.card.valid.standard` that occurs whenever a user unlocks a door using an access card. These two events are in some cases followed by `door.opened` and `door.closed` to register that the door has opened and closed.

Listing 1 shows an event where a door is unlocked using an access card, including three of the most relevant arguments. Namely, the door being accessed, the user accessing it and the access category that the user currently belongs to. The access category correspond to a grouping of users with different door access permissions in the building. This is further discussed in Section 4.5.2.

Listing 2 shows an event where a door is unlocked using an exit button. This type of event does not have an argument containing information about the user since this information is only available from using an access card.

4.3 Events

There are 50 different events in total, all of which are listed in Table A.1 in the Appendix. Table 4.3 displays the relative frequencies of the most relevant events. The relevancy is based on whether they are very common or utilized in upcoming analysis in this chapter. From this table one can see that approximately 97% of the events are

Event Label	Event name	Relative frequency (%)
UE1	access.card.valid.standard	27.95
-	door.requestToExit	26.24
-	door.opened	21.58
-	door.closed	21.58
UE2	access.card.invalid.door	0.126
UE3	access.card.invalid.standard	0.063
UE4	access.card.invalid.inhibited	0.011
-	Remaining 43 events	2.52

Table 4.3: Event distribution. There are 50 types of events in total. Only the relevant events are shown. The user events have been labeled for easier reference.

represented by only four events. Note that these four events correspond to doors being opened or closed in the PAC system. The majority of the remaining events are administrative in one way or another, such as registering new users and doors in the system.

Furthermore, the four events starting with `access.card` involve using an access card which consequently means that there is a user associated with it. These events have been labeled for easier reference later in the thesis. The most common user related event is `access.card.valid.standard` and corresponds to the access card being valid for the given door, and the door should subsequently open. The other three corresponds to the access card being invalid for the given door, and will not result in the door opening. These user related events will be used when finding subgroups of users that behave similarly.

4.4 Door Analysis

Table 4.4 shows all the door names and a corresponding acronym for easier reference later in the thesis. It also shows the total references from any user related event in the log data for each door. From this, it is clear that about six of the doors correspond to a great majority of the log activity.

Figure 4.1 shows the door access distribution for each of the 4 dominant events from Table 4.3. From this one can see that the events `door.opened` and `door.closed` are only logged for the office door (`OD`) and the staff entrance (`SE`). However, the summed frequency of the events `access.card.valid.standard` and `door.requestToExit` do not correspond to the frequency of `door.opened` and `door.closed`. The reason for this is unknown but may be due to the doors being opened and closed in ways other than as a consequence of the events `access.card.valid.standard` or `door.requestToExit` occurring. Additionally, the table reveals that the fire door (`FD`) has a large number of events corresponding to `access.card.valid.standard`, but no events corresponding to `door.requestToExit`. This may indicate that it is only possible to open this door from one direction, or that unlocking the door with an access card is required in both directions.

User Events		
Door Acronym	**Door Name**	**References**
PD	Production Door	103,757
SE	Staff Entrance	83,000
FD	Fire Door	42,822
OD	Office Door	34,918
MG	Main Gate	20,770
ME	Main Entrance	10,123
HR	HR Office	1,780
POA	Payroll Office A	1,244
KC	Key Cabinet	1,129
POB	Payroll Office B	1,018
FO	Finance Office	921
G	Gate	529
PA	Payroll Archive	73
SG	Sliding Gate	58

Table 4.4: Door reference distribution across all user related events. There are 14 doors in total.

It should be noted that only knowing which user is responsible for about 28% of the events is a potential limitation. The reason is that the full movement of each user can not be tracked since the access card is only required in one direction for most of the doors.

4.5 User Analysis

As mentioned in Section 4.2, each of the four user related events listed in Table 4.3 (UE1, UE2, UE3 and UE4) have a number of features associated with them in the original access logs. In order to make the user analysis more manageable, the user events were filtered out from the access logs. Additionally, only a subset of the available features were kept for further analysis. These features are summarized in Tabl 4.5. The filtered data is then saved in a file following a CSV format. The resulting dataset has 302,144 rows in total, where the rows correspond to all the events sorted in a chronological order. The data matrix is denoted $\boldsymbol{X}$ and has dimensions $302,144 \times 6$.

4.5.1 User Movement

Figure 4.2 visualizes the distribution of the average number of events per day for each user. There are 671 different users recorded in the access log data. One can see that most users have about 2-4 events per day on average. The average number of events in a day across all users is 2.8. However, 73 of the users have an outlying average according to the box plot.

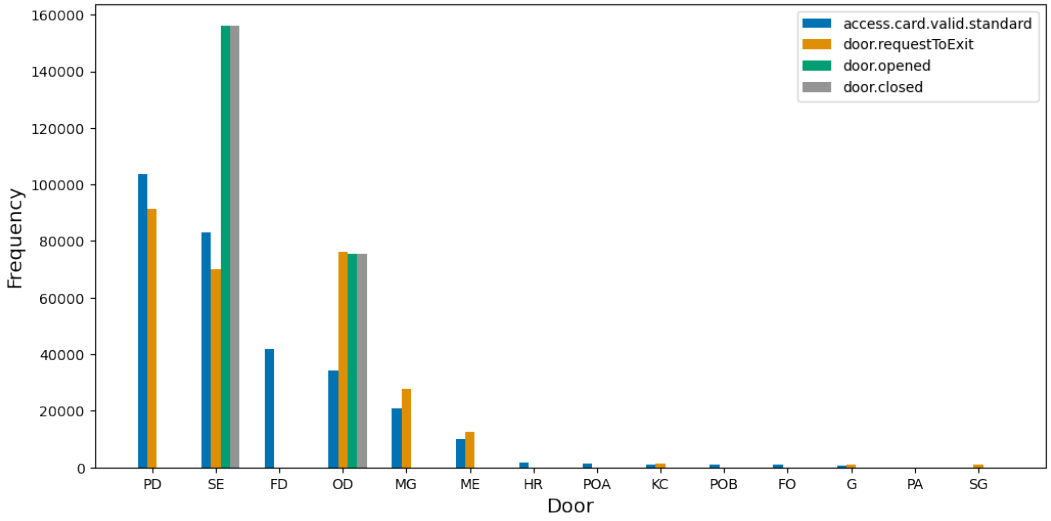


Figure 4.1: The door access frequency for the four dominant events.

Feature	Data type	Feature description
Timestamp	String	Timestamp of the event.
User ID	Integer	Unique identifier of the user.
Door ID	Integer	Unique identifier of the door being accessed.
Door	String	A descriptive name of the door being accessed.
Access category ID	Integer	A unique identifier indicating which access category the user belongs to.
Access category	String	A descriptive name of the access category.

Table 4.5: Description of features extracted from the user related events.

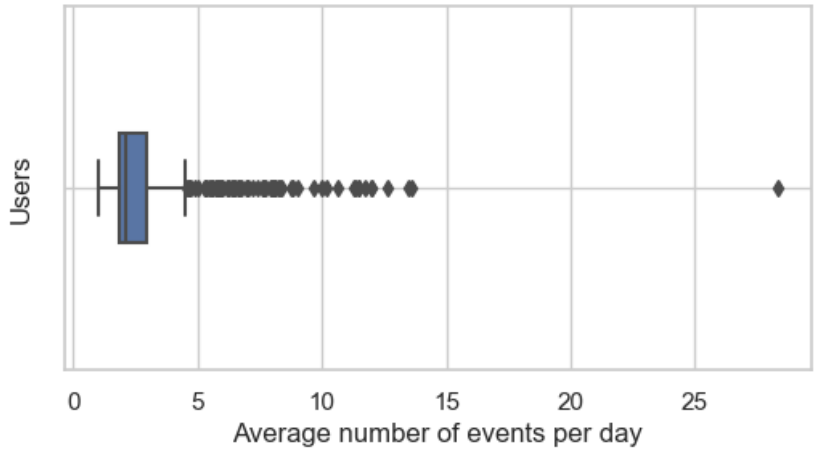


Figure 4.2: Visualization of the distribution of the average number of events per day for each user. The average number of events in a day across all users is 2.8. 73 out of 671 users have an outlying average.

Figure 4.3 shows the frequencies of the 20 most common door access sequences taken in a single day. The purpose of this plot is to illustrate what sort of sequences are common across an entire day for the users. The sequence {`SE - PD`} is the most common sequence by far. This makes sense since these are also the two most accessed doors individually, as seen in Table 4.4. Entering the staff entrance and subsequently opening the production door is therefore the most common movement in a day. This also shows that most users have no other events apart from those initial events at the start of their shift. There are no events when they leave since they can open doors from the inside by the press of a button, which triggers the `door.requestToExit` event, which has no user associated with it. The second most common sequence is to just open the staff entrance. However, it is unlikely that a user only walks through the staff entrance in one day which means that subsequent door accesses were simply not recorded in the log data. This can occur if, for example, another user opens the door using their card and simply lets other users walk through without using their own access cards. This ultimately illustrates one of many limitations with this sort of data. The reason why it is a limitation is because the whole idea is to learn about common user behavior, and if this behavior is distorted by such events then the data can never be fully trusted. However, one can see that the occurrence of the proper full sequences are more common than the broken sequences which means that useful patterns can still likely be identified.

The seventh most common sequence is {`ME - OD`}. This indicates that people working in the office enter the building from the main entrance, whereas the people that use the production door enter the building through the staff entrance.

Figure 4.4 shows frequencies of the 20 most common door access sequences of length two regardless of the day. This means that sequences that include events from two different days are counted. As already mentioned, {`SE - PD`} is the most common

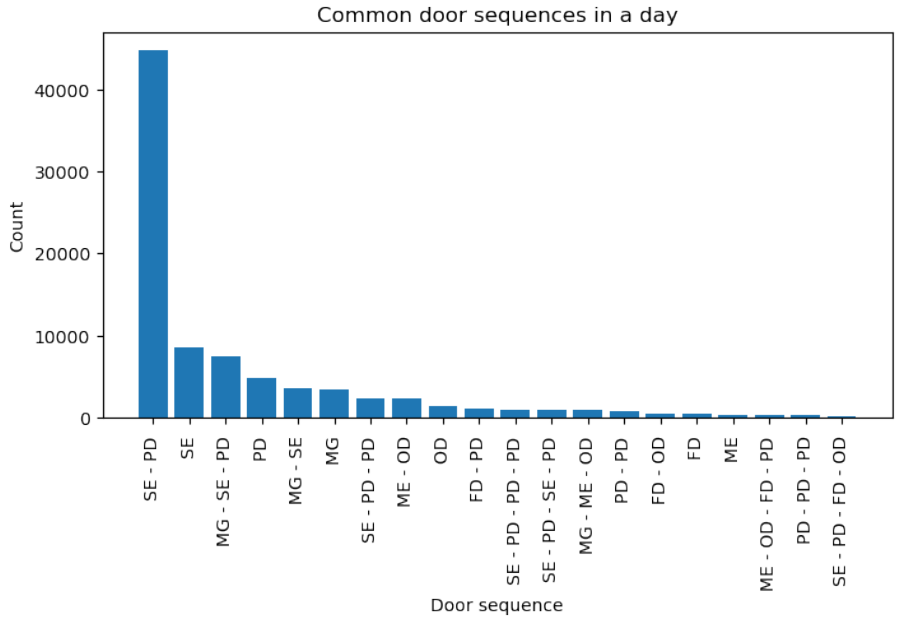


Figure 4.3: Most common door access sequences of any length from the same day. The 20 most common sequences are shown. There are 7,417 unique sequences taken in one day across all users in total.

sequence in one day. The sequences {`SE - PD`} and {`PD - SE`} being the two most common sequences regardless of the day therefore makes sense. The main reason for including this figure is because these are the sequences that will be learned by the machine learning models. The reason for this and exact details of how this will work is explained in Chapter 5.

It is likely common for people to walk the same subsequences across a day multiple times. This can occur since employees may leave the company building and come back later for various reasons. The 25 most common subsequences of lengths 2, 3 and 4 across a day are therefore shown in Figures 4.5-4.7. In order to visualize some of the sequences that include doors with low frequencies, the subsequence counts are normalized by the least common door in the sequence. Sequences of lengths longer than 4 are not shown because these sequences generally consist of subsequences of length 4 or shorter. In other words, from manual inspection it seems that sequences of lengths 4 and below show all interesting sequences that users take.

Figures 4.5-4.7 provide a number of insights regarding the overall movements in the building. One of the various gates (`MG`, `G` and `SG`) are often accessed prior to one of the entrances (`ME`, `SE`). As already mentioned, `ME` is generally followed by `OD`, whereas `SE` is generally followed by `PD`. In most cases, after accessing `PD` the user does nothing more in that day. Users accessing `OD`, however, also tend to subsequently access one of the other office related doors (i.e., `HR`, `POA`, `POB`, `FO` or `PA`). Finally, one can note that the fire door (`FD`) is used to move between `PD` and `OD`. This proves the earlier theory that the fire door can indeed be opened from both directions using an access card (i.e., no request to exit button).

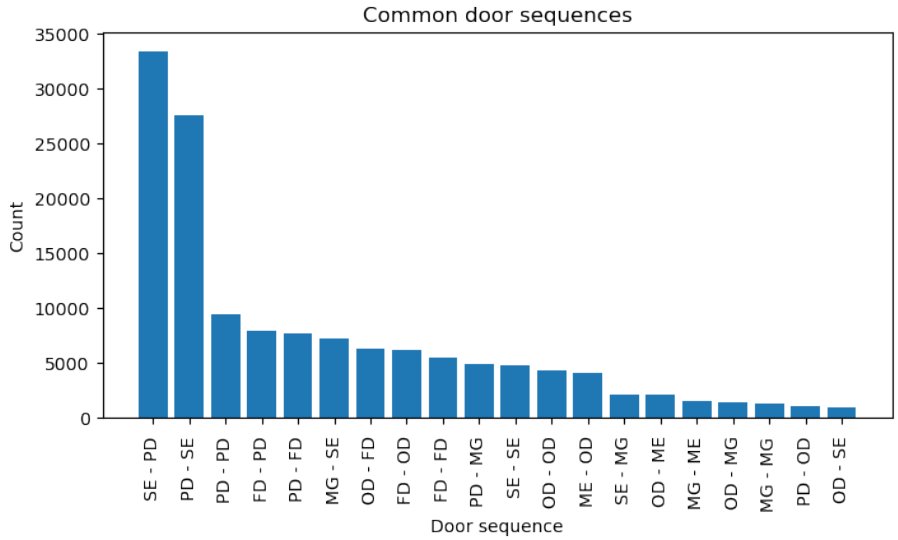


Figure 4.4: Most common door access sequences of length two regardless of the day. The 20 most common sequences are shown. There are 149 such unique sequences across all users in total.

Given all of this information, Figure 4.8 displays an approximate layout of the building. The numbers correspond to how many times two door accesses have occurred in sequence. Note that only the most important paths are included in this figure. There are other paths recorded in the log data that are not visualized. The main reasons for not including them is that these paths do not make sense (e.g. walking through `OD` before `ME`) or that they are very uncommon.

4.5.2 Access Category

The main purpose of the data analysis is to discover subgroups of users that tend to behave similarly (as will become clear in Chapter 5). The access category is a natural place to start since this is a predefined categorization of user privileges inside of the company building. Table 4.6 lists all access categories, their corresponding frequency in the log data and introduces acronyms for easier reference later in the report. Table 4.7 shows the door access distribution of user related events for each access category. As a reminder, the event `UE1` corresponds to a user trying to access a valid door. The events `UE2`, `UE3` and `UE4`, on the other hand, correspond to a user trying to access an invalid door.

The category `FA` has a very low frequency and manual inspection revealed that it was only used for administrative purposes during the initial setup phase of the PAC system. Because of this, the remaining analysis of access categories will not include `FA`.

For the remaining access categories one can draw a number of conclusions regarding which doors are accessible for each category. Let $\boldsymbol{D}$ denote the set of all doors from Table 4.4. Furthermore, let $\boldsymbol{D}_i$ denote the accessible doors for access category i.

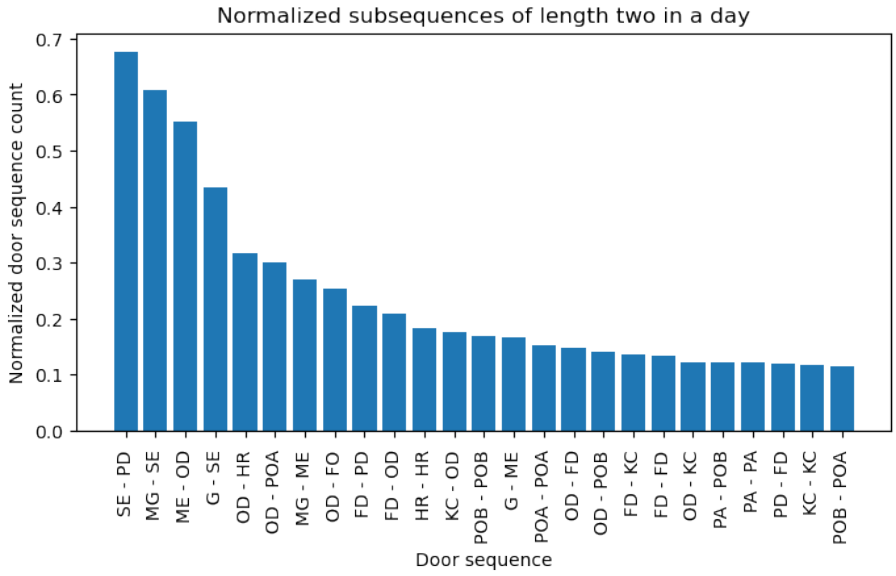


Figure 4.5: Most common door access subsequences of length two from the same day. The 25 most common subsequences are shown. There are 148 subsequences of length two in total.

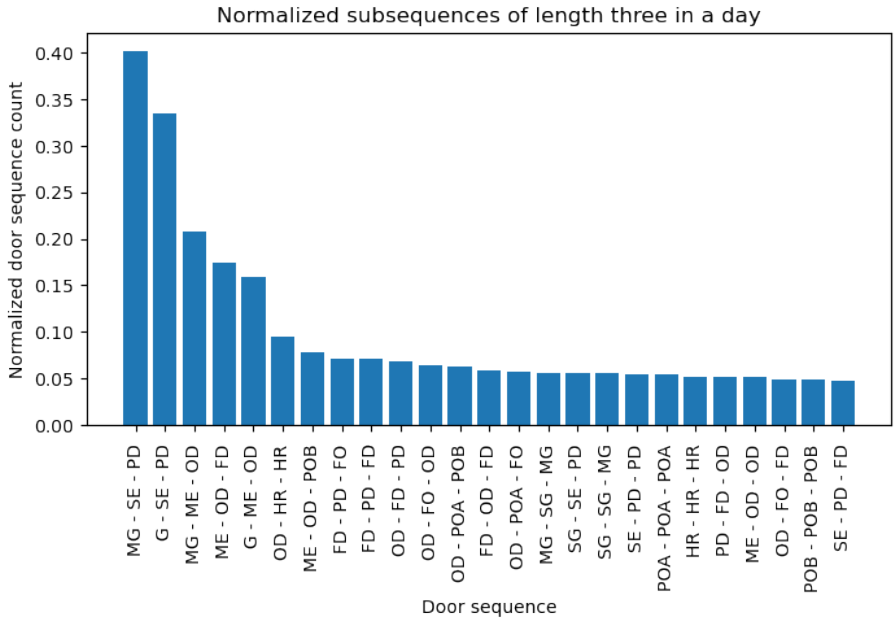


Figure 4.6: Most common door access subsequences of length three from the same day. The 25 most common subsequences are shown. There are 532 subsequences of length three in total.

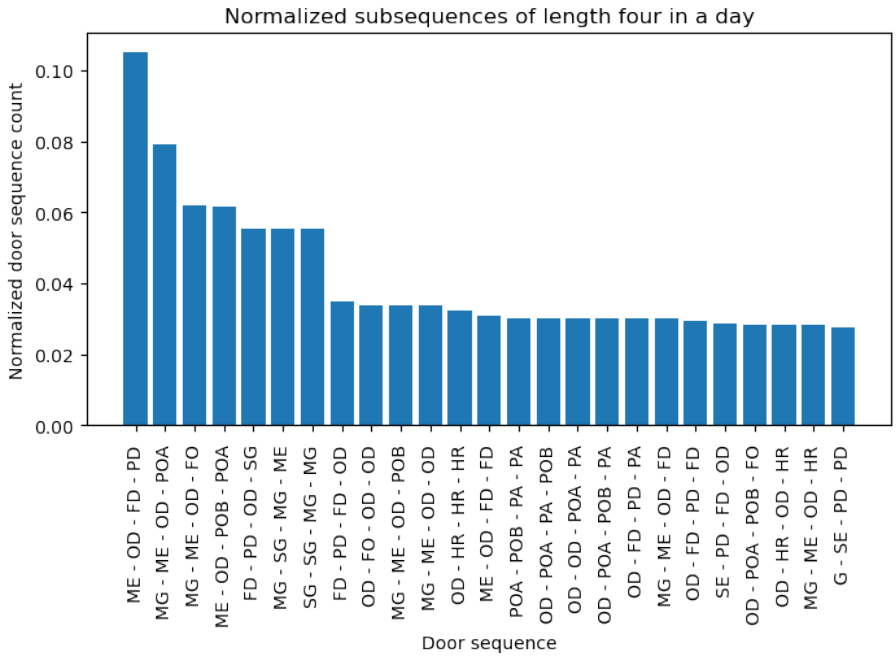


Figure 4.7: Most common door access subsequences of length four from the same day. The 25 most common subsequences are shown. There are 1,272 subsequences of length four in total.

Access Category Acronym	Access Category	Frequency	No. Users
AC3	Access category 3	152,116	257
AC1	Access category 1	125,270	455
AC4	Access category 4	21,671	27
AC2	Access category 2	3,051	15
FA	Full access	36	3

Table 4.6: Distribution of access categories across all user related events. Note that the number of users in each cluster does not sum to the total of unique users in the system (which is 671). The reason for this is because some users have events from more than one access category (i.e., they have had their access category changed at some point).

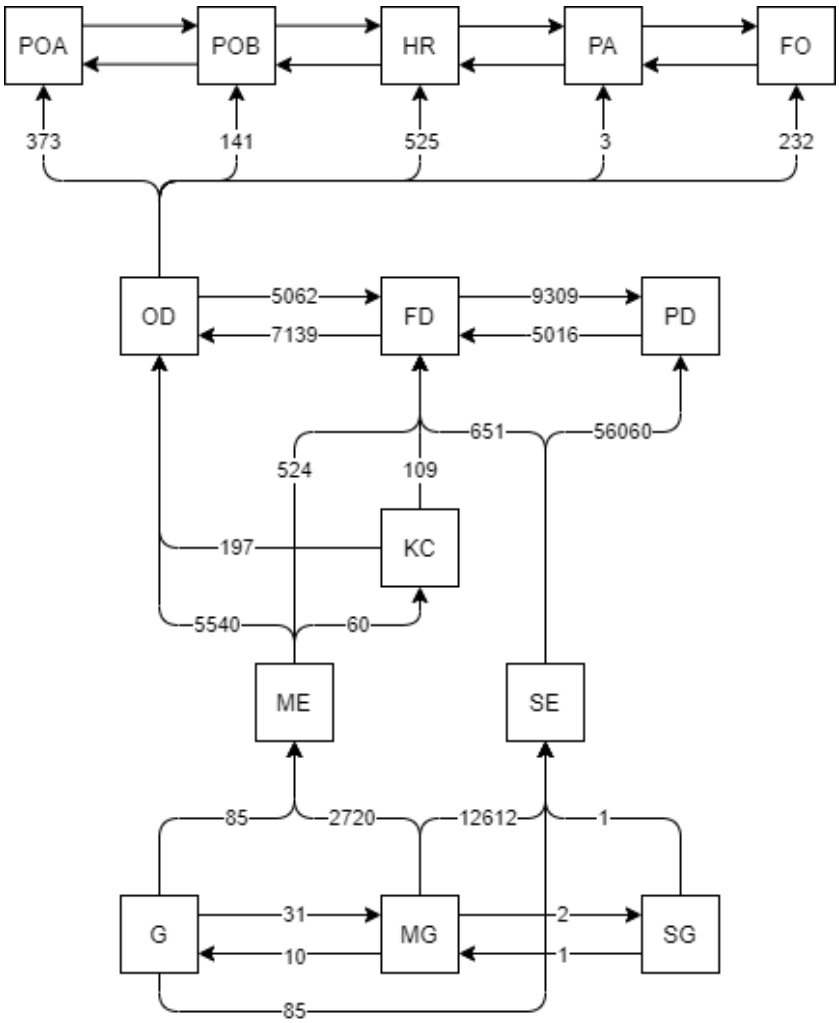


Figure 4.8: An approximate layout of the building only based on analysis of the log data. The numbers correspond to how many times two door accesses have occurred in sequence.

Given this, the sets of accessible doors for each access category can be defined as shown in Eq. 4.1.

$$\begin{aligned} \boldsymbol{D}_2 &= \{\texttt{PD}, \texttt{SE}, \texttt{MG}, \texttt{G}, \texttt{SG}\} \\ \boldsymbol{D}_1 &= \boldsymbol{D}_2 \setminus \texttt{SG} \\ \boldsymbol{D}_4 &= \boldsymbol{D} \\ \boldsymbol{D}_3 &= \boldsymbol{D}_4 \setminus \texttt{SG} \end{aligned} \tag{4.1}$$

From the above one can conclude that AC1 and AC2 access the same doors, except that AC2 also have access to SG. Similarly, AC3 and AC4 access the same doors, except that AC4 also have access to SG.

There are two additional important things to note regarding Table 4.7. The first is that the same access category have accesses from multiple user events for the same door. For example, AC1 has 55,856 accesses with event UE1 and 11 accesses with event UE4 for the door PD. This seems contradicting considering the opposite meaning of UE1 and UE4. One theory for why this occurs could be that AC1 has restrictions that are not limited to the accessible doors, but also to the hour and weekday. However, from manual inspection, there is no indication that the access categories are limited to certain hours or weekdays. The reason for this is because all access categories have events of type UE1 on all hours and weekdays. Instead, it seems that UE4 corresponds to a door being temporarily invalid for a particular access category. This means that administrators can temporarily restrict access of particular doors that are normally allowed for each respective access category. For cases when certain doors of a particular access category have events of types UE1 and UE2 or UE3 the reasons must be different. In most of these cases the occurrence of UE2 or UE3 are very low compared to UE1. From inspection it seems that these occurrences came before correct permissions had been given to each respective access category. This is one suggested reason but note that there is no way to guarantee that this is the true reason based only on the access logs. Additionally, there may be other reasons that were not discovered in the analysis made in this project.

The second thing to note is that the door G was assigned to $\boldsymbol{D}_2$ despite AC2 having no valid accesses for this door. The reason for this is because it also had no invalid accesses, indicating that no one from AC2 has ever tried opening G. This means that one can impossibly know whether this door is allowed or not purely based on analysis of the access logs. This is not a serious issue for the remainder of this project, considering that it is only affecting one uncommonly accessed door for one access category. $\boldsymbol{D}_2$ can simply be updated accordingly whenever new access logs are available. A similar thing can be noted for some of the office related doors for both AC1 and AC2. However, since OD is clearly invalid for these access categories, the other office related doors should consequently be invalid too.

		Door													
AC	Event	PD	SE	FD	OD	MG	ME	HR	POA	KC	POB	FO	G	PA	SG
FA	UE1	0	0	12	0	1	0	0	0	0	0	0	4	0	0
	UE2	0	0	0	0	0	0	0	0	0	0	0	12	0	5
	UE3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	UE4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AC1	UE1	55,856	56,498	0	0	11,042	0	0	0	0	0	0	243	0	0
	UE2	0	0	887	0	13	75	0	0	0	0	0	1	0	1
	UE3	0	0	0	542	0	0	0	0	13	0	0	0	0	0
	UE4	11	54	0	0	28	0	0	0	0	0	0	6	0	0
AC2	UE1	1,335	1,446	0	0	207	0	0	0	0	0	0	0	0	9
	UE2	0	0	48	0	0	0	0	0	0	0	0	0	0	0
	UE3	0	0	0	5	0	0	0	0	1	0	0	0	0	0
	UE4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AC3	UE1	42,293	23,904	34,833	28,813	8,281	8,205	1,538	1,219	809	992	517	249	57	0
	UE2	0	0	107	137	0	0	12	1	0	2	2	0	3	34
	UE3	0	0	0	7	0	0	92	0	0	4	0	0	0	0
	UE4	2	1	0	0	2	0	0	0	0	0	0	0	0	0
AC4	UE1	4,259	1,097	6,931	5,406	1,196	1,843	122	19	306	16	400	14	9	9
	UE2	0	0	0	0	0	0	1	5	0	3	2	0	4	0
	UE3	0	0	0	0	0	0	15	0	0	1	0	0	0	0
	UE4	1	0	4	8	0	0	0	0	0	0	0	0	0	0

Table 4.7: Door access distribution of user related events for each access category.

4.5.3 Categorization of User Behavior

A number of statistics that capture the user behavior criteria mentioned at the start of this chapter was extracted for each of the 671 users. These statistics include the number of accesses of all 14 doors, the number of accesses on each of the 7 weekdays and the number of accesses during each of the 24 hours in a day. This means that each user has $14+7+24 = 45$ features that describe their overall behavior. Furthermore, certain users will naturally have more events in the logs due to them being employed for a longer period. The raw frequency of events is therefore not sufficient in terms of categorizing similar user behavior across employees that have worked for different lengths. The frequencies of each user are therefore normalized by the total events for the given user. After all data preparation, a user data matrix $\boldsymbol{U}$ can be described as shown in Eq. 4.2.

$$\boldsymbol{U} \in [0,1]^{671\times 45} \tag{4.2}$$

Furthermore, the user matrices for each individual feature is also constructed, as shown in Eq. 4.3.

$$\begin{aligned} \boldsymbol{U}_{\text{door}} &\in [0,1]^{671\times 14} \\ \boldsymbol{U}_{\text{weekday}} &\in [0,1]^{671\times 7} \\ \boldsymbol{U}_{\text{hour}} &\in [0,1]^{671\times 24} \end{aligned} \tag{4.3}$$

Each of the 671 rows (users) should now be categorized into different groups (clusters) that behave similarly. In order to visualize potential clusters, two dimensionality reduction techniques were applied to each of the user matrices above, namely PCA and t-SNE. Figure 4.9 plots two components of PCA and t-SNE for each of the four

cases. The clusters from $\boldsymbol{U}_{\text{door}}$, $\boldsymbol{U}_{\text{weekday}}$ and $\boldsymbol{U}_{\text{hour}}$ are presented to ensure that each feature has some impact on the grouping of user behavior. Additionally, the access category of each user is highlighted in the plots with different colors and shapes. Note that 73 of the users have had their access category changed at some point, resulting in them having events with different access categories. For these users, a new category named `Combination` is shown in the plots. Effectively all of those 73 cases correspond to the user having their access category changed from `AC1` to `AC3`. This means that they are granted more permissions within the building.

From these plots it is clear that t-SNE is finding more well defined clusters, indicating the nonlinear structure of the patterns in higher dimensional space. For this reason, the discussion will mostly be with regards to the t-SNE plots. The clearest separation of access categories can be seen in plot 4.9 (d), which is based on $\boldsymbol{U}_{\text{door}}$. This is logical considering that the access categories are by definition a categorization of accessible doors. Furthermore, the users belonging to `AC1` and `AC2` tend to cluster together and users belonging to `AC3` and `AC4` tend to cluster together. This of course makes sense since their door access permissions are practically the same. One can also note that there are additional clusters within the access categories, indicating that there are different behavior within each access category as well.

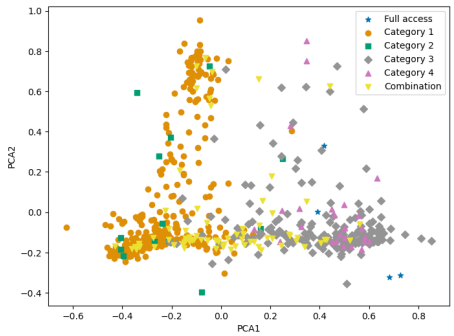
Additionally, the weekdays and doors also seem to have an impact as clusters can indeed be identified in the plots 4.9 (f) and 4.9 (h) as well. For these cases, especially the weekdays, the separation of access categories are expectedly not as clear. The clusters based on $\boldsymbol{U}$ are shown in plot 4.9 (b), and are quite clear as well. It is of course these clusters that are the most interesting as they correspond to users that behave similarly with respect to all criteria.

The users that have had their access category changed are in certain cases placed in their own cluster somewhere in between the other main clusters. The reason for this is likely because these users have access logs that correspond to two different behaviors. The conclusion from this is that when separating the access logs into different behavior, that separation should not be done based on all events from each user, but a separation of events based on their access category for each user may instead be appropriate. This is further discussed in the next chapter. A further analysis of what behavior is common in each of the clusters shown in Figure 4.9 is purposely left out, as such analysis will be presented in the next chapter.

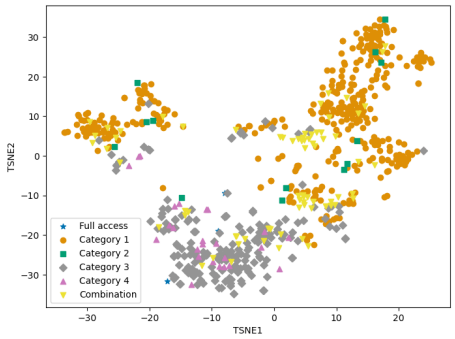
4.6 Log Analysis Conclusions

This section concludes a few important findings from this chapter. Each finding is listed and discussed in the list below.

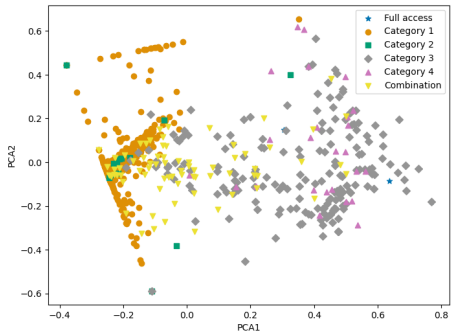
- **Limited knowledge of user behavior**. As already mentioned, only 28% of the events track which user caused them. These are the events when a user need their access card to enter a room. From a security perspective this makes sense since entering a new room is generally more sensitive than leaving a room



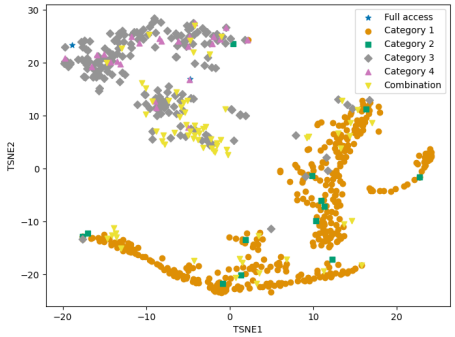
(a) All features (PCA)



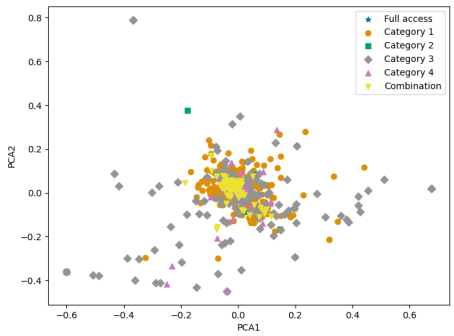
(b) All features (t-SNE)



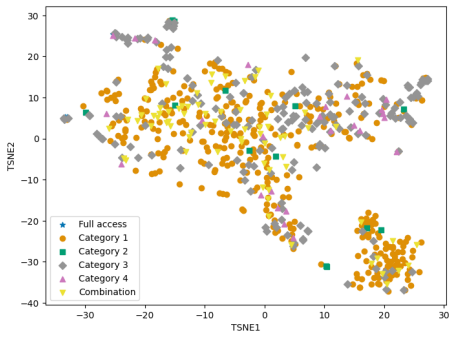
(c) Doors (PCA)



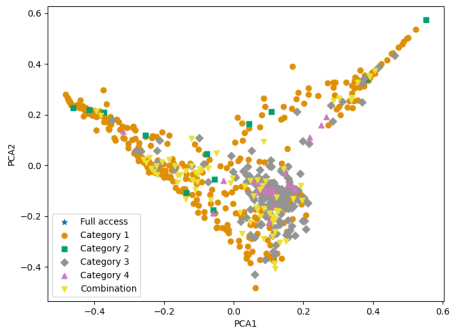
(d) Doors (t-SNE)



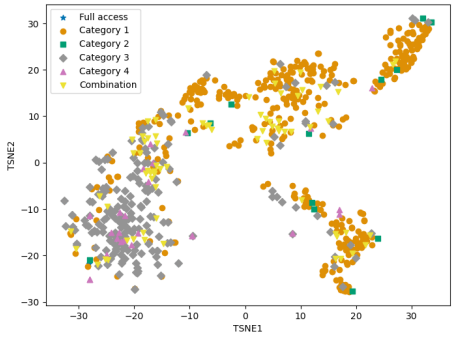
(e) Weekdays (PCA)



(f) Weekdays (t-SNE)



(g) Hours (PCA)



(h) Hours (t-SNE)

Figure 4.9: The two most significant components plotted for PCA (a) and t-SNE (b). The dimensionality reduction is based on 45 original user features. Additionally, the access category of each user is visualized in the plots.

that you were already in. However, being aware of the exact movement of every user (including when they leave doors) can likely help when trying to detect unusual behavior. This is especially true for more complex building layouts. The building layout in this project, however, is quite simple which means that sequences only corresponding to new rooms being entered is sufficient.

- **Broken sequences**. The previous issue is inherent to the PAC system itself. Another problem mentioned briefly in Section 4.5.1 is when parts of sequences are missing. For example, if a user walks the sequence {`SE - PD`}, but only the access to `SE` was logged. This problem is instead caused by misuse of the PAC system. In other words, one user may open a door using their access card and then allow other people to walk in behind them without using their own access cards. According to Bodforss Consulting AB, it may even happen that a door is remained unlocked for an unknown period of time, consequently allowing anyone to enter it without an access card. This allows for unauthorized users to enter disallowed rooms. For example, someone from `AC1` may enter `OD` without it being logged. If it is not logged then the anomaly detection system (see Chapter 5) can never detect it. Furthermore, these distorted sequences may be seen as noise in the data that do not represent the true sequences that should be learnt by the anomaly detection system. This may consequently make it more difficult for the anomaly detection system to report proper anomalies. From a security perspective this is an obvious issue. The solution to this problem is not very straight forward, however. The obvious approach is to enforce stricter rules to make sure that the users use the PAC system as intended by, for instance, making them aware of why the PAC system is there in the first place.

- **Access categories**. The access categories are user privileges defined within the PAC system. Four such access categories were presented, namely `AC1`, `AC2`, `AC3` and `AC4`. From closer inspection it was revealed that `AC1` is very similar to `AC2` and `AC3` is very similar to `AC4`. In fact, as will be explained in more detail in the next chapter, the categories that are similar will be merged into one category, leaving only two categories to consider.

- **Clustering of user behavior**. Section 4.5.3 showed that there are groups of users that behave similarly with respect to which doors they access and when they access them (i.e., the hour of the day and day of the week). However, it also showed that certain users have events corresponding to different access categories, meaning that their door access permissions changed in the span of the logs. This also means that the behavior of these users (which is defined by their events in the logs) changed over time. A proper way of finding clusters is therefore to first split the data by access category, and then separately cluster these datasets by user. The reason for doing this is because it ensures that each cluster will contain events that follow a consistent behavior throughout the full time period. This makes it easier for the anomaly detection system to understand what is normal and what is not. This will

be utilized in Chapter 5 when splitting the users into clusters. Note that this was only necessary because of the fact that certain users have had their access categories changed. Additionally, in another PAC system where access categories may not be present, the clustering would be done normally, by following the approach in Section 4.5.3 without any split on access categories.

5
Method

The previous chapter presented the analysis of the physical access logs. This chapter describes how an anomaly detection framework was developed partly based on the conclusions from the previous chapter. Section 5.1 begins by motivating a split of the users into subgroups that behave similarly. Subsequently, the same section presents how the subgroups were found and finally visualizes their differences in behavior. Section 5.2 describes the architecture used for detecting anomalies in the physical access log data, namely LSTM autoencoders. Based on the subgroups of users found and the chosen anomaly detection architecture, Section 5.3 presents the framework developed for detecting anomalies in real-time. The framework involves appropriate data preparation, an offline learning phase, a real-time online anomaly detection phase and finally analysis of detected anomalies. Another aspect of the project is to provide explanations for why the machine learning models consider a particular sequence of events to be anomalous. Section 5.4 therefore explains how this was achieved for the chosen machine learning architecture. Finally, Section 5.5 describes how the anomaly detection capability of the framework is evaluated.

5.1 Clustering of Users

Eventually a machine learning model will be trained to learn the different user behavior in order to detect abnormal activity. There are three possible approaches to achieve this, all of which are listed below.

- **Training one model for all events**. The drawback of this method is that the model will have to learn too many different patterns. Moreover, it will be heavily biased by the most common types of behavior, i.e., users walking the door access sequence {`SE - PD`}. This means that the model may start believing that this sequence is normal for all users, while in reality it might not be.

- **Training one model per user**. The drawback of this method is that certain users have too little data to learn from. When the model is deployed for practical usage, there may even be entirely new employees entered into the system that have no data at all. In this case there would be no model at all for them to be assigned to. Additionally, this method will mean that each model will believe that the behavior that is common for the given user is what is normal. This approach is clearly wrong since the majority of behaviors from a single user may or may not be malicious. Therefore, normal behavior should in general be de-

Dataset	No. events	Start date	End date
Training set	241,714	2018-02-14	2020-01-21
Test set	60,430	2020-01-21	2020-08-26

Table 5.1: General information regarding the training and test set.

cided based on common overall behavior, and not the behavior from single users.

- **Training one model per cluster of users**. This approach, which is taken in this project, finds a middle ground that tries to minimize the problems with the two first approaches. The data will be separated into k parts that correspond to events that follow similar patterns. Using this approach the bias of overly common patterns are reduced, and there will still be sufficient data to train one model per cluster. Additionally, new users entered in the system can be assigned to a cluster based on pre-existing knowledge of the user, such as their access category. Note that the choice of k is important as this is used to balance between the previously mentioned drawbacks. One model per user corresponds to $k = 671$, since there are 671 users and $k = 1$ corresponds to one model for all the data.

5.1.1 Clustering

The original full dataset $\boldsymbol{X}$ is split into a training set (80%) and a test set (20%). There are therefore two new datasets $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{X}_{\text{test}}$. Some general information regarding these datasets are summarized in Table 5.1. All the clusters presented in this section are only based on $\boldsymbol{X}_{\text{train}}$. $\boldsymbol{X}_{\text{test}}$ can then be accurately used for testing the anomaly detection capabilities of the architecture described in the next section.

It should be noted that even for $k = 671$ the bias would not be completely eliminated. The reason for this is because the behavior of the same user might change over time, effectively causing a dataset shift which was explained in Section 3.3.4. This can be mitigated by only training the models on the most recent data. Section 7 further discusses this issue. For the data in this project, the access categories assigned to each event can be utilized to mitigate this issue to some extent, as was hinted in the previous section. A change of access category signifies a change of permissions for the user in the building, which will likely correspond to new doors being accessed. Because of this, the 241,714 events in the dataset $\boldsymbol{X}_{\text{train}}$ will be split up into two parts. The first part $\boldsymbol{X}_1$ will contain all events corresponding to access catgegories `AC1` and `AC2`. The second part $\boldsymbol{X}_2$ will contain all events corresponding to access catgegories `AC3` and `AC4`. This means that the users that have had their access category changed at some point will have their events split up into both of these datasets. $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ contain 430 and 267 users, respectively. Note that these do not sum to the total number of unique users, which is 671.

The next step is to construct the user data matrices for $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. This will be done in an identical fashion to how the user matrix $\boldsymbol{U}$ was constructed in the

Cluster	No. events	AC1	AC2	AC3	AC4	No. Users
1	67,227	65,933	1,294	0	0	311
2	40,297	39,016	1,281	0	0	119
3	103,768	0	0	86,661	17,107	215
4	28,683	0	0	28,397	286	52

Table 5.2: General information regarding each of the four clusters.

previous chapter. The user matrix for the events in $\boldsymbol{X}_1$ is described in Equation 5.1.

$$\boldsymbol{U}_1 \in [0,1]^{430\times 45} \tag{5.1}$$

The user matrix for the events in $\boldsymbol{X}_2$ is described in Equation 5.2.

$$\boldsymbol{U}_2 \in [0,1]^{267\times 45} \tag{5.2}$$

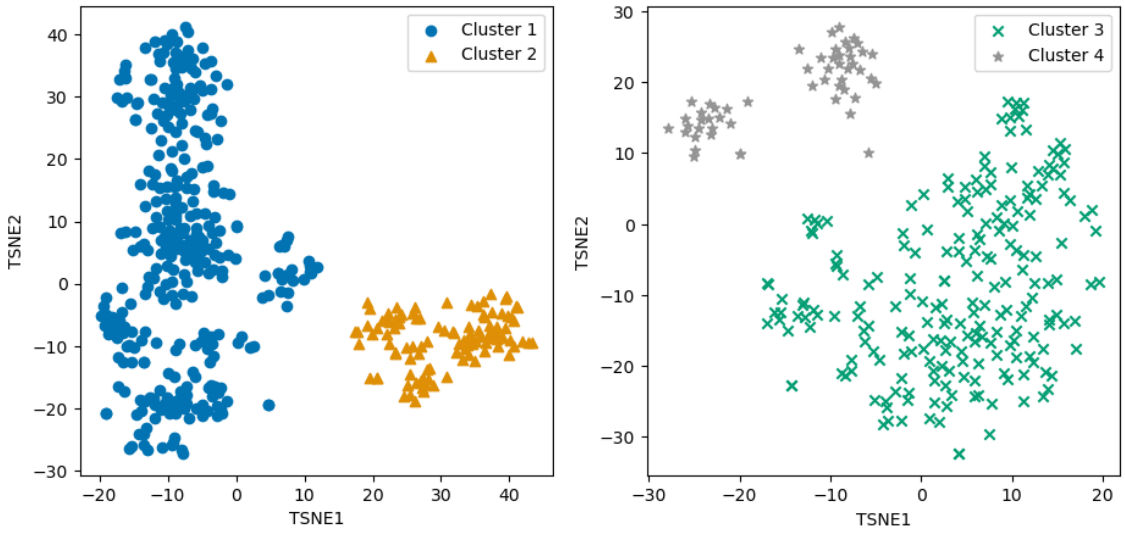
Both PCA and t-SNE was performed on these user matrices. Only the t-SNE plots are presented in the next section, as they gave the best visualizations.

5.1.2 Visualization of Clusters

Figure 5.1 (a) and (b) show the t-SNE clusters for $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$, repsectively. The clusters shown were identified using agglomerative hierchical clustering (see Section 3.4.1). In both cases you can see two clear separations with one larger and one smaller cluster of users. In (b) the smaller cluster is split up into another two clusters. However, for simplicity the value of k (number of clusters) is kept relatively low and the two smaller clusters are therefore put into one cluster. This means that there are four different clusters of users that have been identified and will be further analyzed. Table 5.2 summarizes a few statistics regarding the clusters, including the total number of events, the number of events from each access category and the number of unique users.

Figure 5.2 visualizes the door access distribution for each of the four clusters. The Y-axis corresponds to the mean number of accesses across all users in the corresponding cluster. The plot reveals that the main difference between cluster 1 and 2 is that cluster 2 uses the main gate (`MG`) a lot more. The difference between cluster 3 and 4 is that the users from cluster 3 tend do use the office door (`OD`) and the other doors within the office more frequently. Cluster 4 seems to utilize `SE` and `PD`, which is similar to cluster 1 and 2, but simultaneously uses `ME` and `OD` on occasion like cluster 3.

Figure 5.3 visualizes the weekday access distribution for each of the four clusters. The Y-axis corresponds to the mean number of accesses across all users in the corresponding cluster. From this plot it is clear that the users in clusters 1 and 3 have most of their accesses on normal weekdays. On the other hand, the users in clusters 2 and 4 seem to work on all days of the week including the weekend.



(a) (b)

Figure 5.1: t-SNE based clusters of data with access categories `AC1` or `AC2` are shown in (a). Similarly, t-SNE based clusters of data with access categories `AC3` or `AC4` are shown in (b).

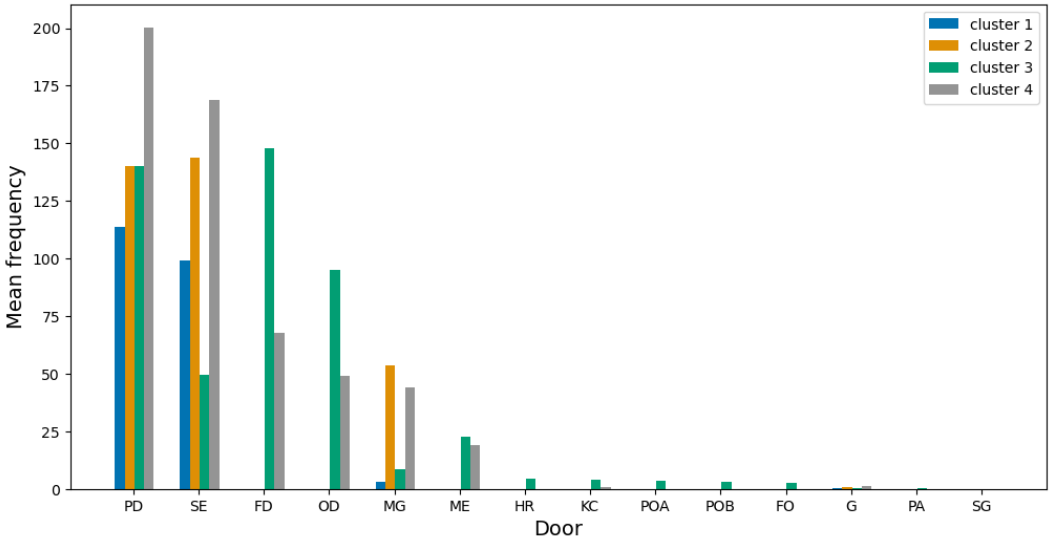


Figure 5.2: Distribution of door accesses for each cluster. The y-axis correspond to the mean across all users in the corresponding cluster.

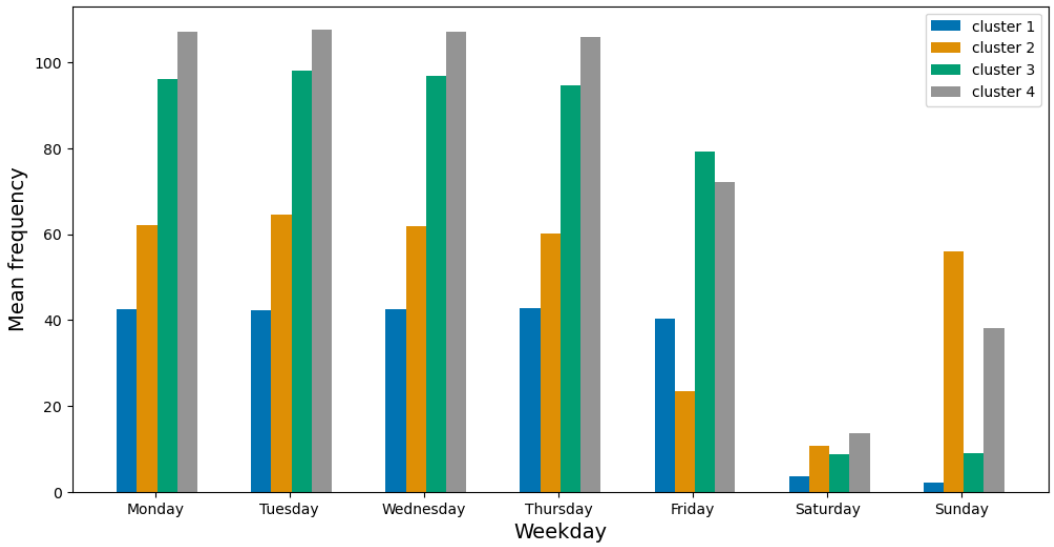


Figure 5.3: Distribution of weekday accesses for each cluster. The y-axis correspond to the mean across all users in the corresponding cluster.

Figure 5.4 visualizes the hourly access distribution for each of the four clusters. The Y-axis corresponds to the mean number of accesses across all users in the corresponding cluster. As was mentioned in Section 4.5.1, most users expectedly have most of their events at the beginning of their shift. The taller bars in the plot are therefore likely to correspond to the starting hours. In this plot it is clear that clusters 2 and 4 contain users that work the night shifts while the users from clusters 1 and 3 dominate the day shifts. However, cluster 4 also has some activity during the day.

Bringing all of these plots together it seems that clusters 2 and 4 work the night shifts and more on the weekend, which seems to have a connection with using the main gate (`MG`) more frequently. Cluster 1 and 3 on the other hand tend to work more normal hours and not so much on the weekend, and perhaps as a consequence uses the main gate less.

5.2 LSTM Autoencoder Architecture

As was explained in Section 3.4.5, autoencoders is an unsupervised machine learning technique with anomaly detection capabilities. The purpose of this project is to find anomalies in an unlabeled dataset which therefore make autoencoders a suitable technique. Furthermore, a number of the anomalies listed in Table 4.1 consist of abnormalities in sequences of events (collective anomaly), and not necessarily from a single event (point anomaly). As explained in Section 3.4.4, LSTM networks are suitable for dealing with data consisting of sequence based features. Because of this, a LSTM autoencoder architecture will be utilized as it fits well for the purposes of this project.

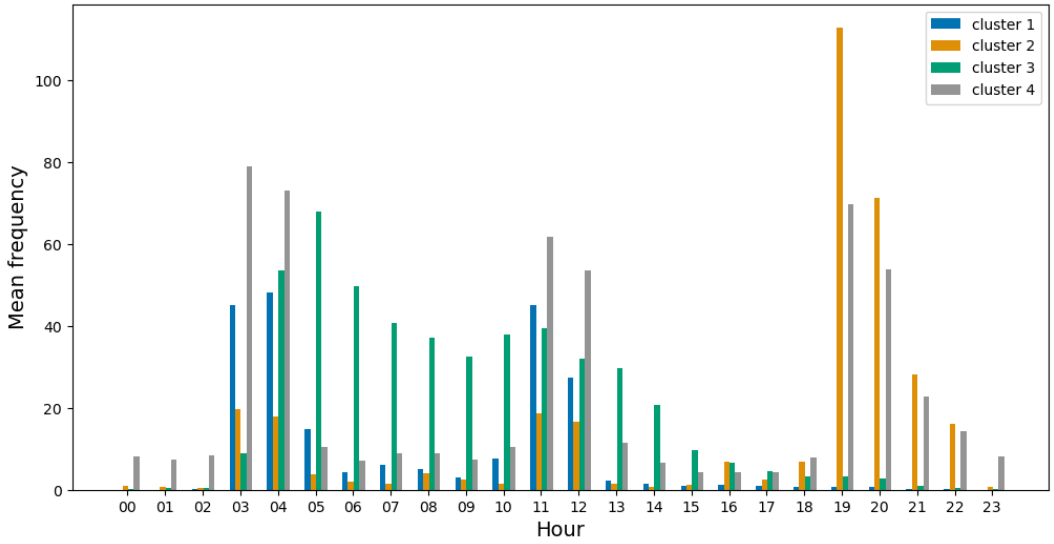


Figure 5.4: Distribution of accesses at all hours of the day for each cluster. The y-axis correspond to the mean across all users in the corresponding cluster.

Figure 5.5 visualizes the details of the architecture used for all trained models. The encoder accepts a sequence of n events $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$ as input and consists of an LSTM network of two stacked layers. The hidden states of the top layer has a lower dimension than the intial layer such that the original inputs are compressed into a lower dimensional representation. The hidden state of the last cell in the top LSTM layer is used as the compressed representation of the original input sequence. The goal of the decoder is to reconstruct the original input sequence based on this compressed representation. It does this by repeating the compressed representation that was output by the encoder n times. This sequence is then fed into another LSTM network. This network consists of two stacked LSTM networks, where the hidden states of the cells in the top network are twice as large as the bottom network. The hidden state from each of the LSTM cells in the top network are then output into a feedforward neural network (FFNN) that in turn outputs the reconstructed sequence $\{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, ..., \hat{\boldsymbol{x}}_n\}$. Exact details of how this is applied to anomaly detection in the context of this project is further discussed in the next section.

As can be seen from the figure, there are a number of hyperparameters to choose such as the sequence length n and the number of units in each of the hidden states h. The choice of these hyperparameters is also discussed in the next section.

5.3 Anomaly Detection Framework

Figure 5.6 visualizes the anomaly detection framework developed in this project. The first step was to identify clusters of different user behavior. This step was presented in Section 5.1, where four different clusters were identified. The next step is the offline learning for each of the four clusters. This step is further explained in

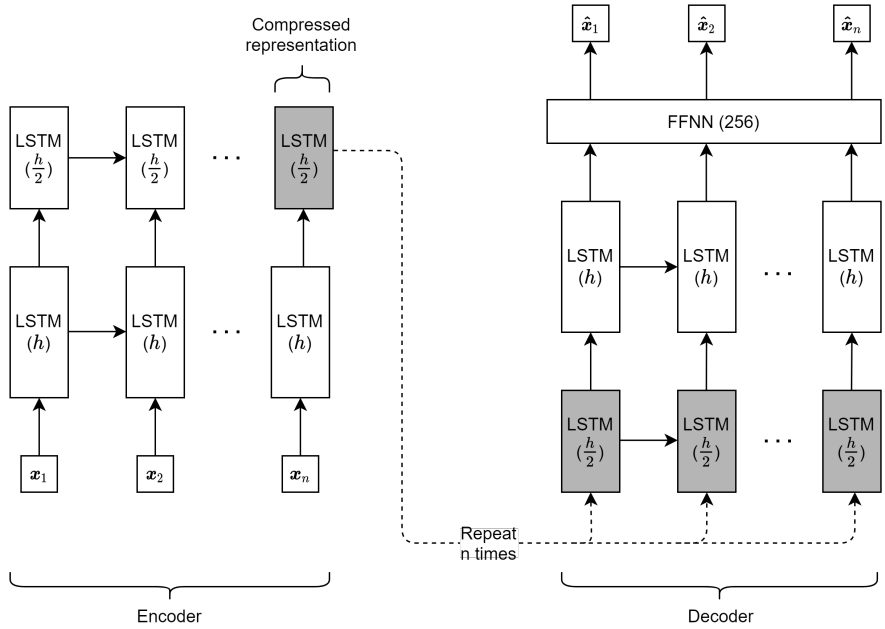


Figure 5.5: A visualization of the autoencoder architecture used to train each of the models. Numbers in parantheses correspond to the number of units in the hidden states of the different networks.

Section 5.3.1. The next steps are to apply the learned models for the purpose of detecting unusual behavior (anomaly detection) and to subsequently analyze the detected anomalies. This is further explained in Section 5.3.2.

5.3.1 Offline Learning

The offline learning phase corresponds to training various LSTM autoencoder models on the data from each of the four clusters. As already mentioned, some of the anomalies listed in Table 4.1 are point anomalies while others are collective anomalies. This means that utilizing a single LSTM autoencoder per cluster will not be sufficient. Instead, there will be one autoencoder for two different sequence lengths. The sequence lengths used are based on the sequences analyzed in Section 4.5.1, where sequences of lengths 1 and 2 seemed the most interesting. Because of this there will be two LSTM autoencoders trained for $n = 1, 2$ for each of the four clusters. This means that there will be $2 \times 4 = 8$ autoencoders in total. This approach makes it possible to recognize normal behavior in individual events ($n = 1$) and for sequences of events ($n = 2$). Additional models for $n = 3, 4$ is likely interesting as well but was not investigated in this thesis.

Feature Extraction

The original full dataset $\boldsymbol{X}$ has dimensions $302,144 \times 6$. This means that there are $302,144$ events, where each event has 6 features as shown in Table 4.5. In general, artificial neural networks require their inputs to be numerical and lie in the range

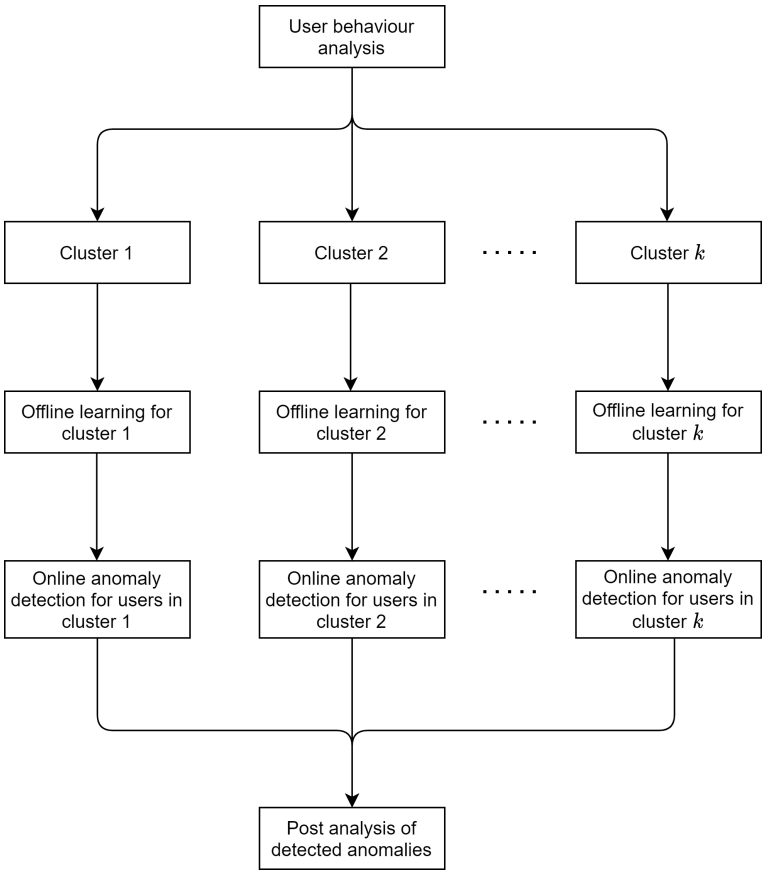


Figure 5.6: A flowchart describing the anomaly detection framework.

Feature	Type	No. categories	Feature description
Time since last event (TSLE)	Numerical	-	Time since the last event occurred. Extracted from the original timestamp feature.
Weekday	Categorical	7	The weekday that the event occurred on. Extracted from the original timestamp feature.
Hour	Categorical	24	The hour of the day that the event occurred on. Extracted from the original timestamp feature.
Door	Categorical	14	The name of the door that was accessed in the event.
Event	Categorical	4	The type of event that was triggered.

Table 5.3: Description of features extracted from the original features shown in Table 4.5.

$[0, 1]$ [15]. A number of numerical features have been extracted from these original 6 features that captures information about the user behavior. These features are meant to be useful when detecting the various anomalies listed in Table 4.1. Table 5.3 lists the extracted features. Note that the feature `Event` is included here because this allows the detection of access card events that lead to a door not opening (see Table 4.3). The training data will only include events of type `access.card.valid.standard` such that the LSTM autoencoder immediately classifies any of the other user events as an anomaly upon employment. This makes sense considering their low frequency compared to `access.card.valid.standard`.

All of the categorical features will be one-hot encoded as described in Section 3.3.6. In short, this means that each category within each of the categorical features will become a feature of their own, and therefore correspond to a column in the data matrix. This means that there will be $1 + 7 + 24 + 14 + 4 = 50$ features in the resulting data matrix. After one-hot encoding the categorical features, they will now lie in the range $[0, 1]$ as desired. The numerical feature `TSLE`, however, is still not in this range. This feature is therefore scaled using a min-max scaler, as described in Section 3.3.6, such that it lies range $[0, 1]$ as well. Note that the scaling was initially done to $\boldsymbol{X}_{\text{train}}$. The same scaler is then used to scale $\boldsymbol{X}_{\text{test}}$. The resulting data matrices is described in Equation 5.3,

$$\begin{aligned} \boldsymbol{X}_{\text{train}} &\in [0, 1]^{241{,}714 \times 50} \\ \boldsymbol{X}_{\text{test}} &\in [0, 1]^{60{,}430 \times 50} \end{aligned} \tag{5.3}$$

The various data preparation steps taken will now be described below. These steps were taken individually for both $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{X}_{\text{test}}$. In order to simplify the notation, $\boldsymbol{X} \in [0, 1]^{N \times 50}$ will be used to describe both $\boldsymbol{X}_{\text{train}}$ and $\boldsymbol{X}_{\text{test}}$.

Data Preparation

The LSTM network of the encoder is what accepts the initial input. In general, LSTM networks accept inputs of dimension $n \times p$, where n is the event sequence length, and p is the number of features used for each event. The dataset $\boldsymbol{X}$ must therefore be split up into sequences of length n. $\boldsymbol{X}$ currently consists of N feature vectors of dimension 50 as shown in Equation 5.4.

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}, \quad \boldsymbol{x}_i \in [0,1]^{50} \tag{5.4}$$

This sequence of feature vectors will be further split up into sequences of length n. Note that only sequences of events taken by the same user are considered. The reason is that sequences including events from different users are very unlikely to follow any interesting patterns since different users generally walk around the building in an uncorrelated fashion. This means that $\boldsymbol{X}$ is further filtered by events by a certain user belonging to one of the four clusters found in Section 5.1. This is described by Equation 5.5,

$$\begin{aligned} \boldsymbol{X}^{(u,\,c)} &= \{\boldsymbol{x}_1^{(u,\,c)}, \boldsymbol{x}_2^{(u,\,c)}, \ldots, \boldsymbol{x}_{N_{u,c}}^{(u,\,c)}\} \\ \boldsymbol{x}_i^{(u,\,c)} &\in [0,1]^{50} \\ N_{u,c} &= \text{Number of events for user } u \text{ in cluster } c \\ c &\in \boldsymbol{C} \\ u &\in \boldsymbol{U}_c \end{aligned} \tag{5.5}$$

where $\boldsymbol{C}$ is the set of clusters $\{1, 2, 3, 4\}$ and $\boldsymbol{U}_c$ is the set of users in cluster c. Splitting $\boldsymbol{X}^{(u,\,c)}$ into sequences can be done in a number of ways. There are three main ways that are worth mentioning. The first is to construct sequences in a sliding-window fashion throughout the entire dataset. A problem with this is that it will include sequences across two different days. For example, the sequence {`PD` - `SE`} will become common (as was illustrated in Figure 4.4 from Chapter 4) since it is common for `PD` to be the last door access of the day, and `SE` to be the first. The second is to use the sliding-window method, but only on events filtered by day. This eliminates the potential issue just mentioned. However, some users work night shifts which means that their events may be split up such that some occur before midnight, while some occur after midnight. A third solution is therefore to construct sequences of events that are at most eight hours from each other. This ensures that sequences only consists of accesses that are made within the same working shift. Despite this, the first method was still utilized in this project. The main reason for this is that despite a sequence like {`PD` - `SE`} not technically being a sequence taken inside of the building, as they are taken on two separate days, are still common occurences in the data. Therefore, it could be seen as another type of user behavior within the data that the autoencoders should pick up on. Table 5.4 describes how these sequences are constructed.

$\boldsymbol{X}^{(u,\,c)}$ is now transformed into $\tilde{\boldsymbol{X}}^{(u,\,c,\,n)}$ with dimensions $m_{u,c,n} \times n \times p$, which consists of these sequences and is described in Equation 5.6.

Seq. number	Sequence
1	$\{\boldsymbol{x}_1^{(u,\,c)},\ \boldsymbol{x}_2^{(u,\,c)},\ \ldots,\ \boldsymbol{x}_n^{(u,\,c)}\}$
2	$\{\boldsymbol{x}_2^{(u,\,c)},\ \boldsymbol{x}_3^{(u,\,c)},\ \ldots,\ \boldsymbol{x}_{n+1}^{(u,\,c)}\}$
3	$\{\boldsymbol{x}_3^{(u,\,c)},\ \boldsymbol{x}_4^{(u,\,c)},\ \ldots,\ \boldsymbol{x}_{n+2}^{(u,\,c)}\}$
$\vdots$	$\vdots$
$m_{u,c,n}$	$\{\boldsymbol{x}_{N_{u,c}-n+1}^{(u,\,,c)},\ \boldsymbol{x}_{N_{u,c}-n+2}^{(u,\,c)},\ \ldots,\ \boldsymbol{x}_{N_{u,c}}^{(u,\,c)}\}$

Table 5.4: Construction of sequences of events using the sliding-window method on the datasets $\boldsymbol{X}^{(u,\,c)}$. Note that $m_{u,c,n} = N_{u,c} - n + 1$ corresponds to the number of sequences constructed for user u in cluster c of length n.

$$\tilde{\boldsymbol{X}}^{(u,\,c,\,n)} = \begin{bmatrix} \boldsymbol{x}_1^{(u,\,c)} & \boldsymbol{x}_2^{(u,\,c)} & \ldots & \boldsymbol{x}_n^{(u,\,c)} \\ \boldsymbol{x}_2^{(u,\,c)} & \boldsymbol{x}_3^{(u,\,c)} & \ldots & \boldsymbol{x}_{n+1}^{(u,\,c)} \\ \boldsymbol{x}_3^{(u,\,c)} & \boldsymbol{x}_4^{(u,\,c)} & \ldots & \boldsymbol{x}_{n+2}^{(u,\,c)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_{N_{u,c}-n+1}^{(u,\,c)} & \boldsymbol{x}_{N_{u,c}-n+2}^{(u,\,c)} & \ldots & \boldsymbol{x}_{N_{u,c}}^{(u,\,c)} \end{bmatrix} \tag{5.6}$$

All the sequences for all users in each cluster is now concatenated into one matrix per cluster per value of n, giving $\tilde{\boldsymbol{X}}^{(c,\,n)}$ with dimensions $m_{c,n} \times n \times p$, where $m_{c,n}$ is the number of sequences of length n in cluster c. These dimensions are appropriate for the LSTM autoencoder input layer. Given that there are four different values of c (four clusters) and 2 different values of n (2 different sequence lengths) means that there are in total $4 \times 2 = 8$ datasets. For each of these datasets, one LSTM autoencoder will be trained.

Learning Process

The parameters of the LSTM autoencoders were optimized using the Adam optimization algorithm. 10% of the training set was used as validation data for the purpose of optimizing various hyperparameters. Table 5.5 presents the chosen hyperparameters. The hyperparameters for each of the 8 models only differed across the value of n, but not for the value of c. The loss value of the validation set was used as metric for deciding these parameters, except for h. The reason why the loss value can not be used as a metric when optimizing h is because the loss value of two different neural networks with a different number of units is not comparable. h was simply chosen such that enough of the patterns could be picked up by respective model. A too low value of h means that a model will underfit the data and a too high value of h means that it will find all possible patterns very easily, including noise (i.e., anomalies) which needs to be avoided. h is higher for $n = 2$ since the data containing sequences naturally contain more patterns. When training an autoencoder for anomaly detection purposes, one would ideally want the training data to not consist of any anomalies. This will naturally make it easier for the model

n	η	θ	**Loss function**	h
1	0.001	0.00001	Smooth L1	16
2	0.001	0.00001	Smooth L1	32

Table 5.5: Summarization of the hyperparameters used. The hyperparameters for each of the eight models only differed across the value of n, but not for the cluster c. η is the learning rate. θ is the weight decay parameter. h is the number of units in the hidden states of the LSTM autoencoders.

to distinguish between what is normal and what is abnormal when tested on the test data. However, considering that the dataset used in this project is completely unlabeled, such information is not available. Therefore, the training set may or may not include anomalies. This is further discussed in the following sections.

The loss function used was a smooth version of the L1 absolute loss. Details of this loss function was presented in Section 3.3.8. The reason for using this loss function is mainly because it is robust to outliers (anomalies) in the data similar to the regular L1 absolute loss. Additionally, the smooth version allows the optimization algorithm to converge more effectively. Being robust to outliers is important for this project considering that potential anomalies in the training data should be ignored as much as possible.

The total training time for all eight models was roughly eight hours. All models were trained on a NVIDIA GeForce GTX 3080 GPU.

5.3.2 Online Anomaly Detection

As already discussed, the purpose of the project is to reduce the amount of manual labor needed by administrators to analyze the very large number of physical access logs. This is done by having the LSTM autoencoders automatically filter out events that correspond to normal behavior. Events that an LSTM autoencoder inteprets as anomalous, on the other hand, will be notified to the administrators for further analysis. The remaining necessity for some manual inspection is important to note since something being anomalous according to an LSTM autoencoder, does not imply harmful behavior. The LSTM autoencoders are simply reporting behavior in new incoming events that are considered unusual based on the behavior learnt from the training data. The decision of whether a reported anomaly is harmful is a far too subjective decision for a machine learning model to make. The flowchart in Figure 5.7 shows the overall workflow of detecting anomalies in the access logs in real-time. The steps of the flowchart are explained further in the numbered list below.

1. Extract relevant features from a new incoming event and prepare it like described earlier in this section in order to retrieve $\boldsymbol{x}_n^{(u,\,c)} \in [0,1]^{50}$.
2. Extract the $n-1$ most recent events for the particular user u for $n = 1, 2$. Now the sequence $\boldsymbol{x}^{(u,\,c)} = \{\boldsymbol{x}_1^{(u,\,c)},\ \boldsymbol{x}_2^{(u,\,c)},\ \ldots,\ \boldsymbol{x}_n^{(u,\,c)}\}$ is constructed for each value of n.

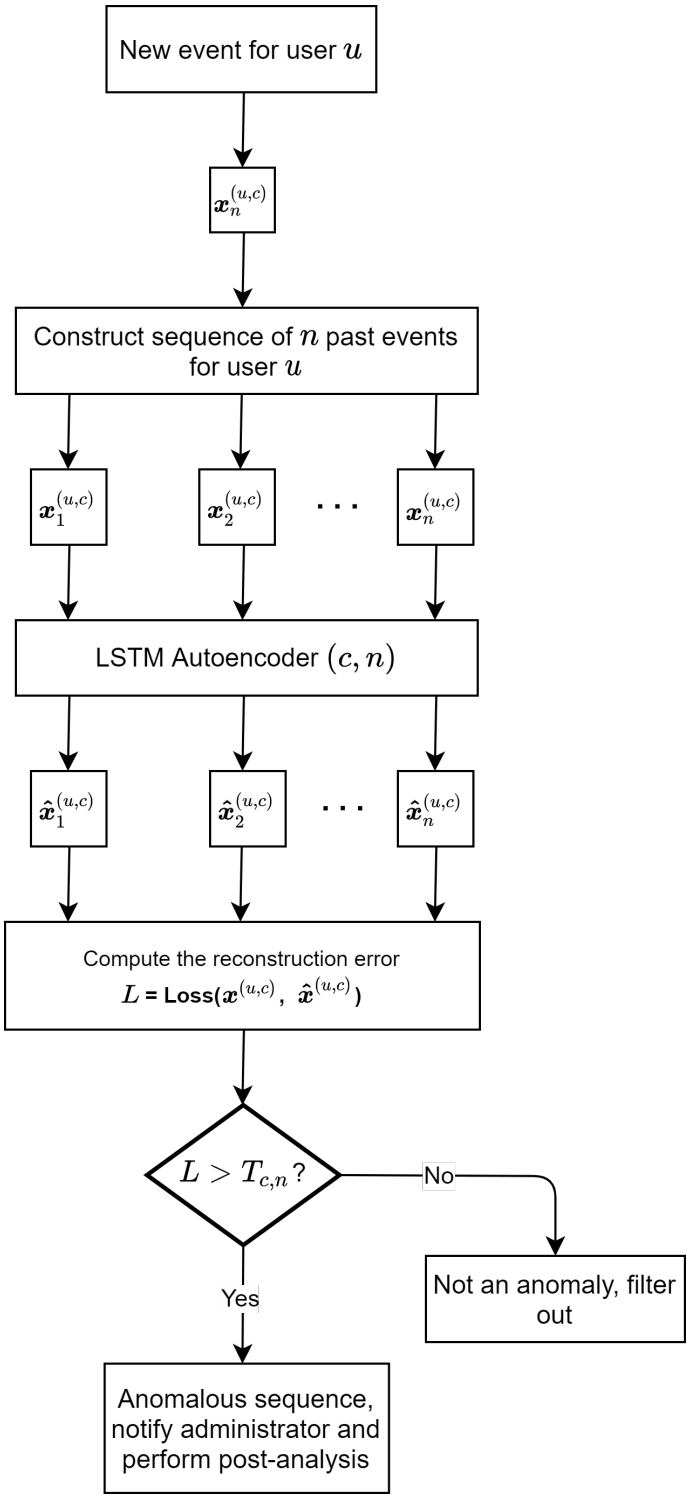


Figure 5.7: Real-time anomaly detection workflow.

3. Retrieve the correct LSTM autoencoder for the corresponding value of n and cluster c that user u belongs to. This retrieval is not always completely straight forward. If the user u has sufficient pre-existing data, then the cluster c will be known and everything works out. However, if the user u is completely new in the system (i.e., no events from the training data), then the cluster for user u must be heuristically decided. One can look at the access category that user u is a member of to narrow it down to two of the four clusters. When deciding between the remaining two clusters additional information about the user's behavior is required, such as whether the user is a night shift worker or not. Another approach is to assign user u randomly to one of the two possible clusters given its access category. After more data has been gathered for this user, one may discover whether this random assignment was correct or not. If it was not, then reassign it to the other cluster for all future checks. Given that correct LSTM autoencoders are chosen, the input sequences are fed into corresponding autoencoder. In turn, the reconstructed sequences $\hat{\boldsymbol{x}}^{(u,\,c)} = \{\hat{\boldsymbol{x}}_1^{(u,\,c)},\ \hat{\boldsymbol{x}}_2^{(u,\,c)},\ \ldots\ ,\ \hat{\boldsymbol{x}}_n^{(u,\,c)}\}$ are output.
4. The reconstruction error L is now computed according to $\mathbf{Loss}(\boldsymbol{x}^{(u,\,c)}, \hat{\boldsymbol{x}}^{(u,\,c)})$, where **Loss** is the L1 absolute loss function.
5. L corresponds to the metric used when deciding whether a particular sequence is anomalous or not. A sequence being anomalous corresponds to L being larger than a predefined threshold $T_{c,n}$. The threshold is subscripted by c and n because each of the 8 datasets naturally require separate thresholds. If $L < T_{c,n}$ then the reconstruction error was low enough to consider the sequence normal. In this case, the sequence is filtered out.
6. If $L > T_{c,n}$ then the sequence can be considered anomalous. In this case, administrators of the system may be notified for further analysis. This analysis could include manual inspection of the sequences in order to understand what caused the anomaly and whether it is harmful or not. In addition to manual inspection one may apply various machine learning explainability techniques in order to understand why the model thought it was anomalous. Section 5.4 will further explain how this was achieved in this project.

Let $\boldsymbol{\Phi}_{c,n}$ denote the set of loss values for all sequences in $\tilde{\boldsymbol{X}}_{\text{train}}^{(c,\,n)}$. The choice of the thresholds $T_{c,n}$ is made based on this set. This choice can be done in many ways. If one is convinced that the training set consists of only normal instances, then all the loss values in $\boldsymbol{\Phi}_{c,n}$ correspond to a normal reconstruction error. In this case one could assign $T_{c,n} = max(\boldsymbol{\Phi}_{c,n})$. Thus, anything larger than the maximum loss value of the training set will be considered anomalous. However, in the case of this project, the training set is unlikely to be free of anomalies. Therefore, one can not assume that the largest value of $\boldsymbol{\Phi}_{c,n}$ is a suitable threshold. Instead, the thresholds are chosen based on $\boldsymbol{\Phi}_{c,n}$ with outliers filtered out. This was done according to Equation 5.7.

$$T_{c,n} = E[\boldsymbol{\Phi}_{c,n}] + \lambda * std[\boldsymbol{\Phi}_{c,n}] \tag{5.7}$$

Thus, a reconstruction error larger than λ standard deviations above the mean of $\boldsymbol{\Phi}_{c,n}$ is considered anomalous. The factor λ multiplied with the standard deviation

was decided empirically. λ is set to 3 for $n = 1$ and 4 for $n = 2$ (across all clusters). The values of the various thresholds $T_{c,n}$, $\forall c \in \boldsymbol{C}$, $\forall n \in \{1, 2\}$ are shown in Table 5.6.

	n	
c	1	2
1	0.1249	0.2666
2	0.2406	0.4108
3	0.7641	0.3163
4	0.9151	0.4860

Table 5.6: Anomaly thresholds $T_{c,n}$, $\forall c \in \boldsymbol{C}$, $\forall n \in \{1, 2\}$.

5.3.3 Discussion of Anomaly Detection

This section will briefly discuss which of the anomalies from Table 4.1 are detectable, and to which degree they are detectable, by the anomaly detection framework. Each anomaly and their detectability is listed below.

- **Door access at an unusual time**. The features `hour` and `weekday` make sure that this anomaly will be identified. Any event that occurs at an unusual time for the given cluster should lead to a larger reconstruction error than usual.

- **First time accessing a door**. The anomaly detection framework only detect unusual behavior with respect to the overall behavior in respective clusters. This means that a particular user accessing a door for the first time will not necessarily be detected. However, if the door being accessed is rare overall for the cluster, then it will be detected. This is not a huge problem since this anomaly can easily be detected from manual checks that do not require machine learning.

- **Denied door access**. A door access being denied will result in any of the events `UE2`, `UE3` or `UE4` occurring. Considering that these events were completely removed from the training set, any occurrence of these in the test set will naturally lead to a high construction error, consequently leading them to be interpreted as anomalies. However, it should be noted that this anomaly does technically not require machine learning as one could simply just track these events.

- **Unusual number of door accesses**. This is the only anomaly that is not detectable by the LSTM autoencoder, but may still optionally be solved with improved accuracy using machine learning. One can get more specific regarding this anomaly by specifying a time interval. For example, an unusual number of door accesses in a day or in a week and so on. A natural approach to this would be to gather the number of door accesses within the given time interval for each user, and then finding statistical outliers. In other words, this anomaly is about outliers in a global statistic of the access logs. The anomalies caught

by the LSTM autoencoder are local to one event or shorter sequences of events.

- **Unusual sequences of door accesses**. This is naturally captured by the chosen LSTM autoencoder architecture. It should be noted again that this thesis only investigated sequences of lengths 1 and 2. Longer sequences may also be useful to detect. Additionally, the model finds unusual sequences only based on the features from Table 5.3.

- **Short time interval between accesses of different doors**. The feature `TSLE` was added specifically to detect this anomaly. This feature ensures that the time between two door accesses is logical.

- **Repeated door accesses**. This can be detected by the LSTM autoencoder for $n > 1$, with the assumption that it is unusual behavior in the training set. However, this anomaly can be detected without machine learning if the intention is simply to detect multiple accesses of the same door without checking that it is unusual behavior.

Conclusively, most anomalies can be detected with the chosen framework. It should be noted that the framework simply detects unusual behavior based on the features from Table 5.3. Since anomalies are by definition about unusual behavior, the framework should theoretically work well. It is of course capable of finding additional anomalies that are not explicitly mentioned above.

5.4 Model Explainability

Deep learning models are known to be difficult to interpret [56]. Since this project utilizes deep learning, the explainability aspect of the project was not completely straight forward. Section 3.3.7 introduced a model-agnostic method for achieving interpretability, namely SHAP (SHapley Additive exPlanations). Model-agnostic methods essentially work by analyzing the impact on the output of a model when changing the input features in different ways. Since deep learning models have complex internals, this approach is appropriate for this project. SHAP is able to give explanations for individual predictions made by a model. In other words, it can explain which features were important when predicting why a sequence of events was anomalous. This is precisely what is needed in this project.

Again, SHAP depends on the model outputting a value, due to SHAP being model-agnostic. This is problematic considering the usage of autoencoders in this project, since autoencoders output a reconstructed sequence. This is not something that SHAP can naturally work with. A solution to this was suggested by Antwarg *et al.* [5]. Their suggestion involves adding an additional layer to the original autoencoder that outputs the reconstruction error directly. This is visualized in Figure 5.8. Note that this layer is only included when retrieving explanations using SHAP, and not while training the original models. Furthermore, outputting the reconstruction error is appropriate as this is a logical metric for deciding which features are important.

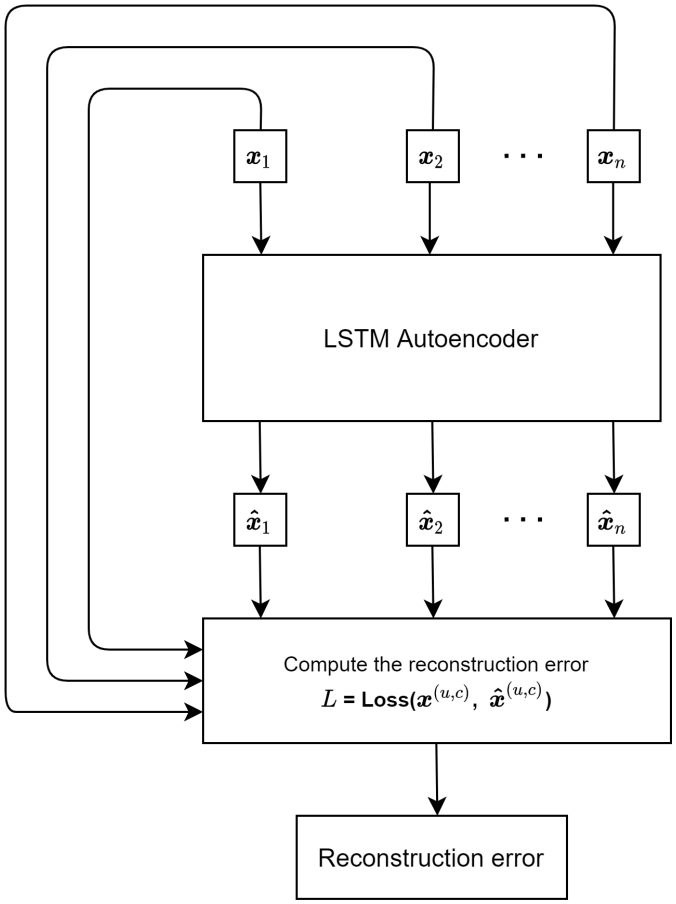


Figure 5.8: Visualization of the addition of a new output layer (i.e., the reconstruction error) such that SHAP values can be computed.

For example, a feature contributing to a higher reconstruction error means that it is likely to have caused the anomaly.

The original paper that introduces SHAP values [4] has provided a Python library with different ways of retrieving SHAP values [57]. The two most relevant methods are called *GradientExplainer* and *DeepExplainer* as they work well for neural network based models. In this project, the GradientExplainer was arbitrarily chosen between the two.

Finally, the SHAP values may be both positive and negative, depending on whether it is contributing to an increase or decrease of the reconstruction error. The minimum and optimal value of the reconstruction error is 0. This means that a feature contributing to a decrease of the reconstruction error is always pushing the instance away from being an anomaly, and vice versa for a feature contributing to an increase. This is because instances with higher reconstruction errors are by definition anomalous. The takeaway is that a feature with a positive SHAP value is *contributing* to the anomaly, while a feature with a negative SHAP value is *offsetting* the anomaly [5].

5.5 Evaluation

In many cases an anomaly detection problem in machine learning consist of a labeled dataset, where it is known which instances are normal and which are anomalous. This makes it a typical binary classification task, where the performance can be objectively evaluated using various classification metrics. The dataset used in this project, however, is unlabeled. This means that it is unknown whether a particular data instance is anomalous or not. Furthermore, it is possible that the dataset consists of only normal behavior. One solution to this could be to perform manual labeling of the dataset. However, this requires a huge amount of effort for a few reasons. First, the dataset is quite large. Second, it is relatively unknown what an anomaly would actually look like.

This project instead resorted to a more subjective type of evaluation. This was done by allowing each of the eight LSTM autoencoders make predictions for their assigned test dataset $\tilde{\boldsymbol{X}}_{\text{test}}^{(c,\,n)}$. This means that there will be a number of normal and abnormal instances. The evaluation will essentially consist of comparing these two types of instances for all the models. This will be done both on a global and a local scale. These two cases are listed and further discussed below.

- **Global evaluation**. This corresponds to presenting the *overall* difference between normal and abnormal instances for each model. For example, which values for each of the five features (i.e., event, door, hour, weekday and TSLE) were typical for normal compared to abnormal instances. Furthermore, by summing the SHAP values across all instances one can get an overall idea of which of the features were the most impactful on the reconstruction error. In other words, which features were the most likely to cause instances to be anomalous.

- **Local evaluation**. This corresponds to analyzing individual instance predictions. This was done by randomly retrieving two normal and two abnormal instances for each of the eight models and then analyzing them. The analysis will be done through manual inspection as well as generating the SHAP value of each feature for the given prediction. The SHAP values will indicate which feature(s) were the most impactful when deciding whether a particular instance was anomalous or not.

The above points are essentially about presenting what each of the eight models are capable of. The actual evaluation will come down to the domain experts from Bodforss Consulting AB carefully inspecting these results and then reporting how practically useful it is.

6
Results

This chapter presents the results. The results are based on evaluations of the found anomalies for each model, as described in Section 5.5. Section 6.1 shows how well each of the 8 models learnt the patterns of respective dataset. Section 6.2 visualizes the various thresholds $T_{c,n}$ relative to the reconstruction error of all instances in respective dataset. Section 6.3 presents the global evaluation. Section 6.4 presents the local evaluation. Discussions of the results presented in the various plots and tables of this chapter is presented in Chapter 7.

6.1 Training Process

Figure 6.1 shows the average loss value across all instances at each epoch for the training and validation set.

6.2 Anomaly Thresholds

The reconstruction errors (loss values) for both the test and training sets for all values of c and n are shown in Figures 6.2 and 6.3. The red vertical line in each plot corresponds to the threshold chosen (according to Eq. 5.7). Note again that the thresholds were only chosen based on the training sets.

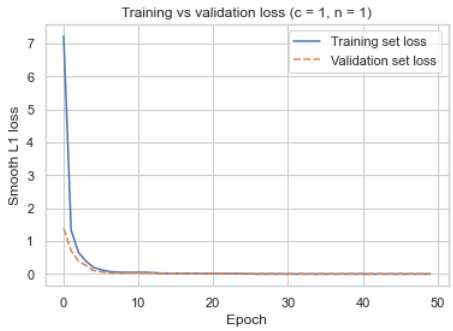
6.3 Global Evaluation

Section 6.3.1 begins by showing the global evaluation of point anomalies ($n = 1$). Section 6.3.2 shows the global evaluation of collective anomalies ($n = 2$).

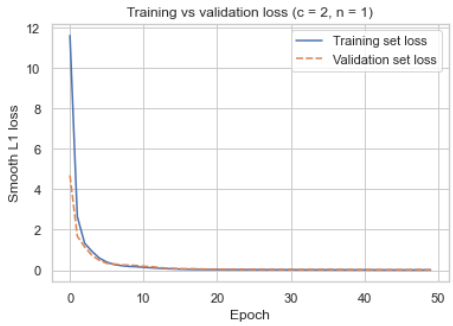
6.3.1 Point Anomalies

Figure 6.4 shows the difference in door accesses between normal (a) and anomalous (b) instances. Figure 6.5 shows the difference in weekdays accesses between normal (a) and anomalous (b) instances. Figure 6.6 shows the difference in accesses at all hours of the day between normal (a) and anomalous (b) instances.

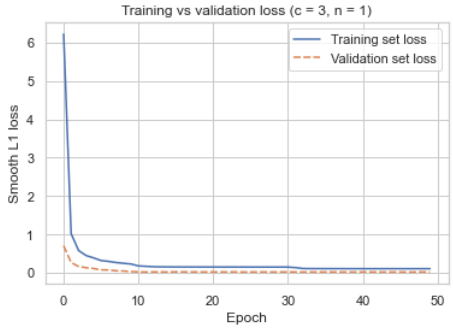
Figure 6.7 shows the distribution of SHAP values for all normal (a) and anomolous (b) instances for each feature in each cluster. Note that there is some variation in the provided methods for computing SHAP values. Because of this, the SHAP values



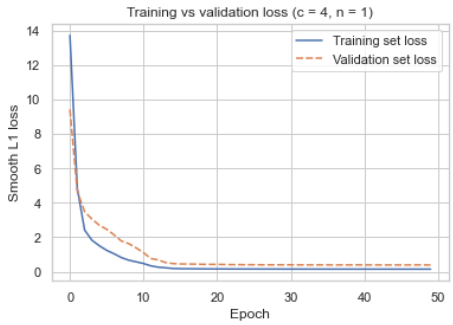
(a)



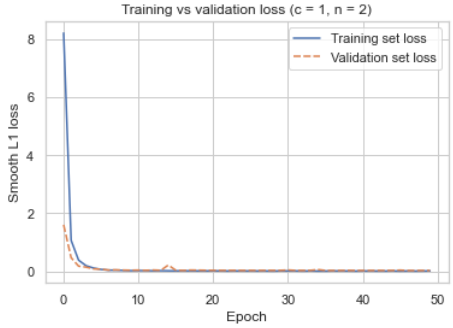
(b)



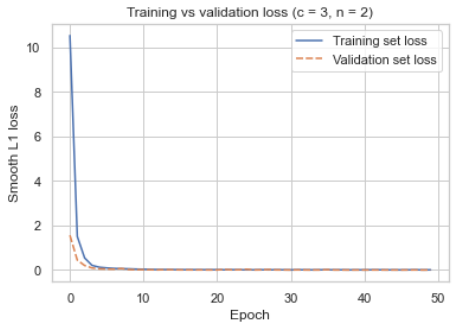
(c)



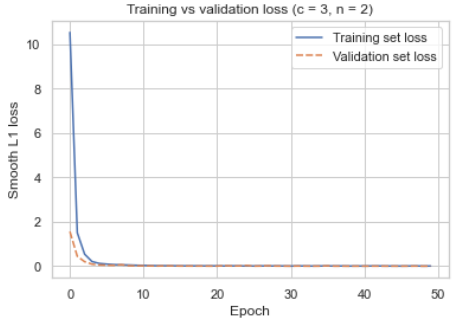
(d)



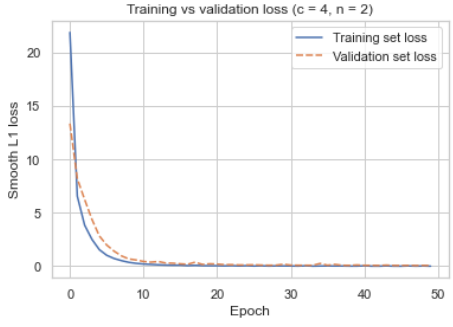
(e)



(f)

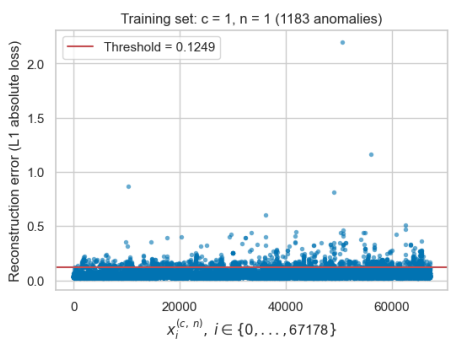


(g)

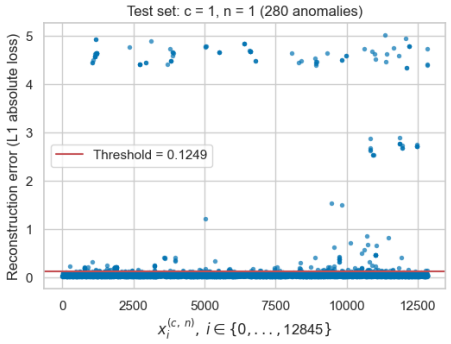


(h)

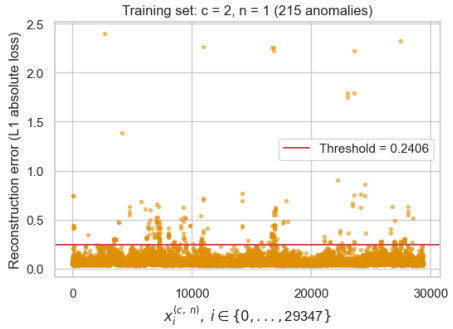
Figure 6.1: Training and validation loss vs epoch plotted for each cluster and sequence length.



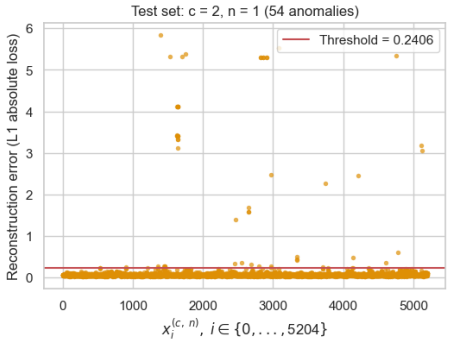
(a)



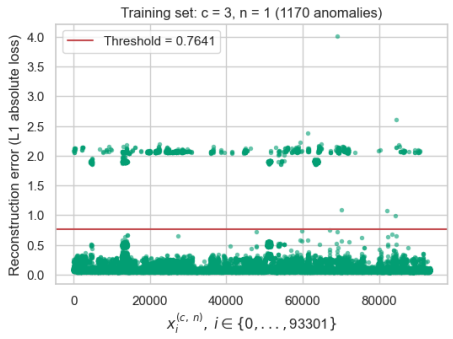
(b)



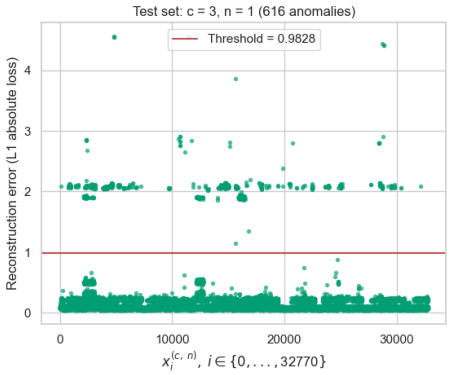
(c)



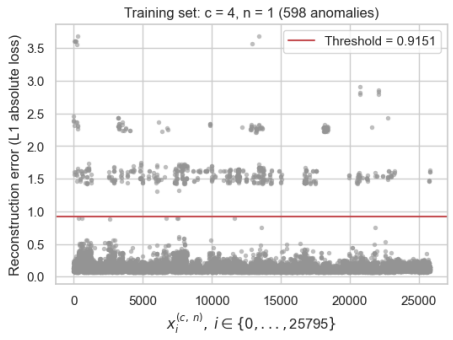
(d)



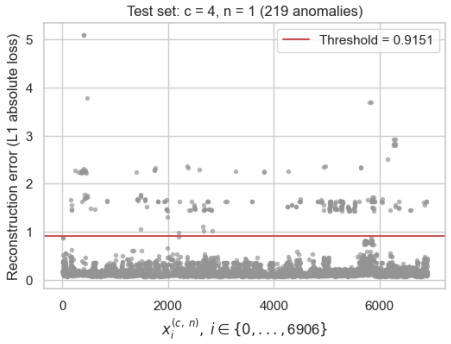
(e)



(f)

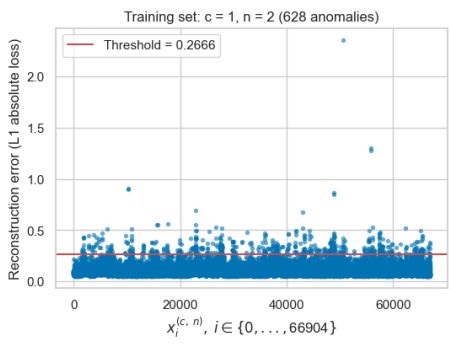


(g)

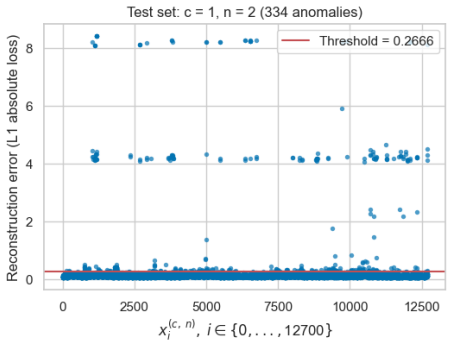


(h)

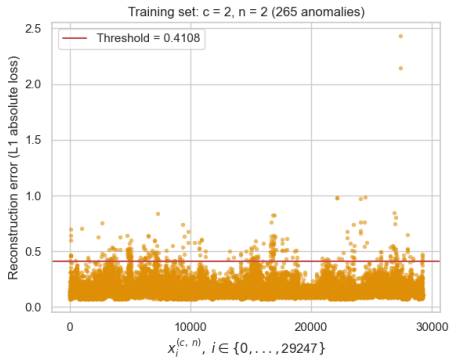
Figure 6.2: Reconstruction error distributions for the training and test sets for $n = 1$.



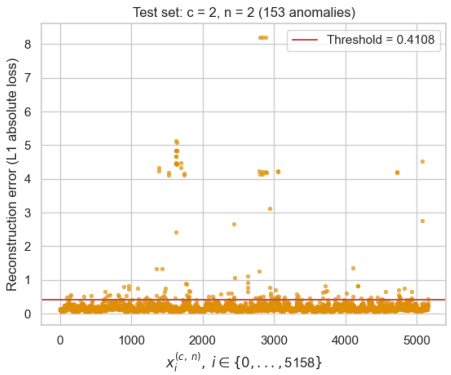
(a)



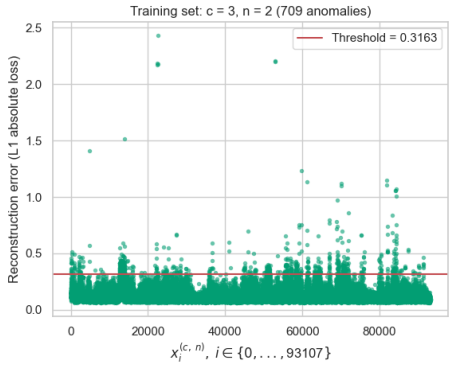
(b)



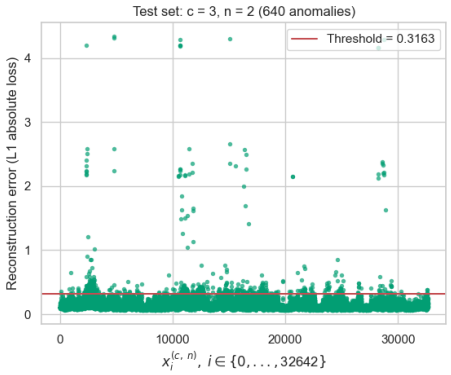
(c)



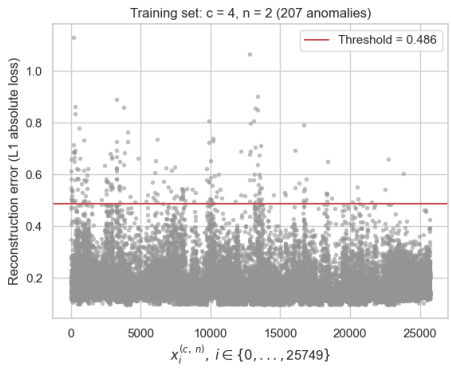
(d)



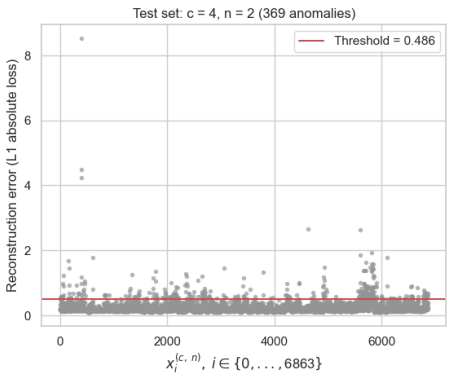
(e)



(f)

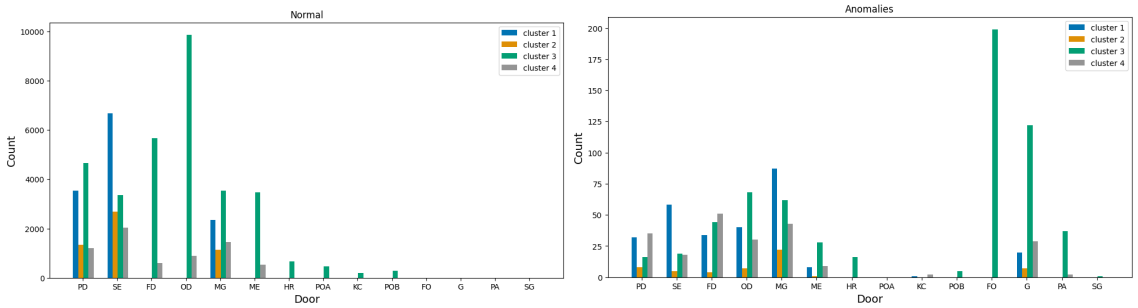


(g)



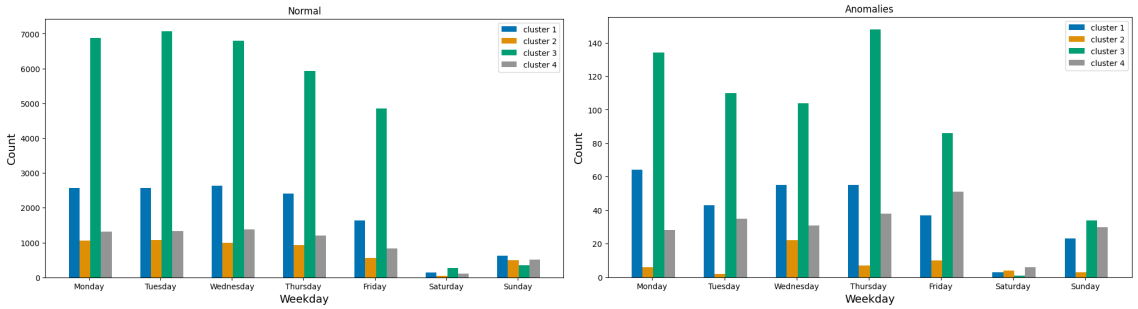
(h)

Figure 6.3: Reconstruction error distributions for the training and test sets for $n = 2$.



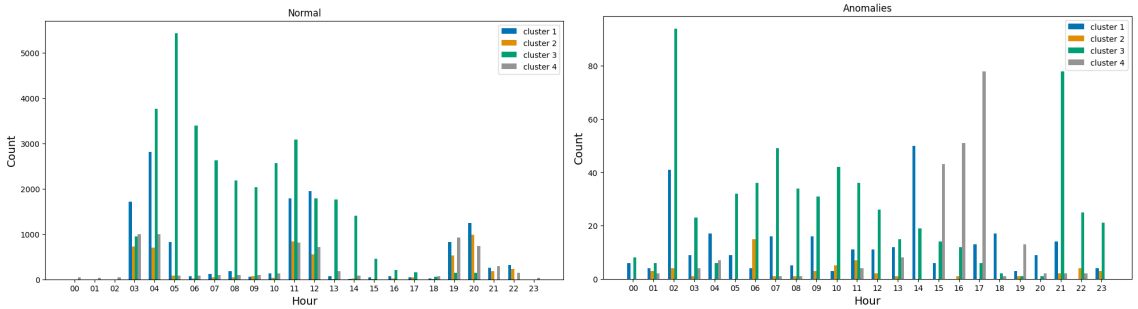
(a) **(b)**

Figure 6.4: Distribution of door accesses for each cluster for normal (a) and anomalous (b) instances. This is based on predictions for $n = 1$.



(a) **(b)**

Figure 6.5: Distribution of weekday accesses for each cluster for normal (a) and anomalous (b) instances. This is based on predictions for $n = 1$.



(a) **(b)**

Figure 6.6: Distribution of accesses at all hours of the day for each cluster for normal (a) and anomalous (b) instances. This is based on predictions for $n = 1$.

shown in this plot correspond to the average of 100 runs.

Figure 6.8 plots the SHAP values together with the reconstruction error for each anomalous instance for each cluster. Note that a corresponding plot for normal instance was purposely left out as these only showed lines around zero. The reason for this is because normal instances have a reconstruction error equal to or very close to zero, which means that the SHAP values will be close to zero as well.

6.3.2 Collective Anomalies

As mentioned in the previous chapter, only collective anomalies corresponding to event sequences of length 2 were considered. Figure 6.9 shows the 15 most common sequences in each cluster for anomalous instances. Figure 6.10 shows the 15 most common sequences in each cluster for normal instances.

Figure 6.11 shows the distribution of SHAP values for all anomalous (a) and normal (b) instances for each feature in each cluster. Again, the SHAP values shown in this plot correspond to the average of 100 runs.

6.4 Local Evaluation

Section 6.4.1 begins by showing the local evaluation of point anomalies ($n = 1$). Section 6.4.2 shows the local evaluation of collective anomalies ($n = 2$).

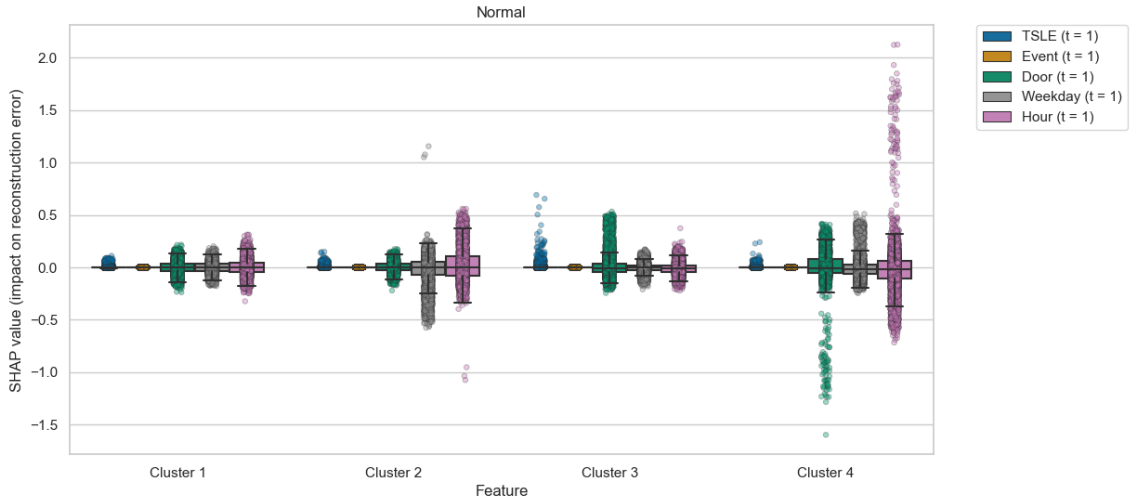
6.4.1 Point Anomalies

Table 6.1 summarizes the chosen instances that were used for local evaluation for $n = 1$. Two anomalous and two normal instances were chosen for each cluster.
Figures 6.12-6.15 show the distribution of 500 retrievals of the SHAP values of each feature for all anomalous and normal instances for each cluster.

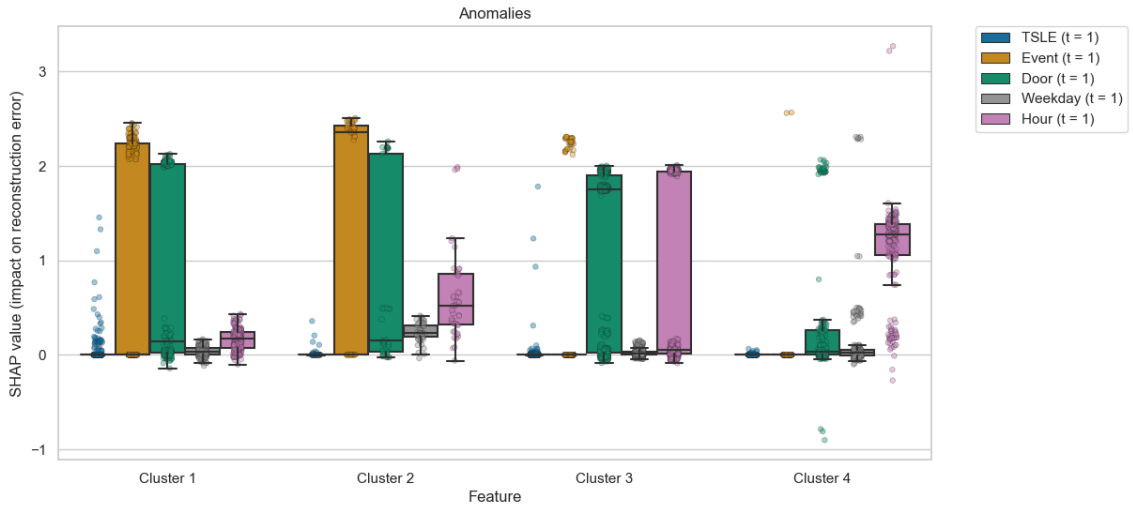
6.4.2 Collective Anomalies

Table 6.2 summarizes the chosen instances that were used for local evaluation for $n = 2$. Two anomalous and two normal instances were chosen for each cluster.

Figure 6.16 shows the distribution of 500 retrievals of the SHAP values of each feature for all anomalous instances for each cluster.

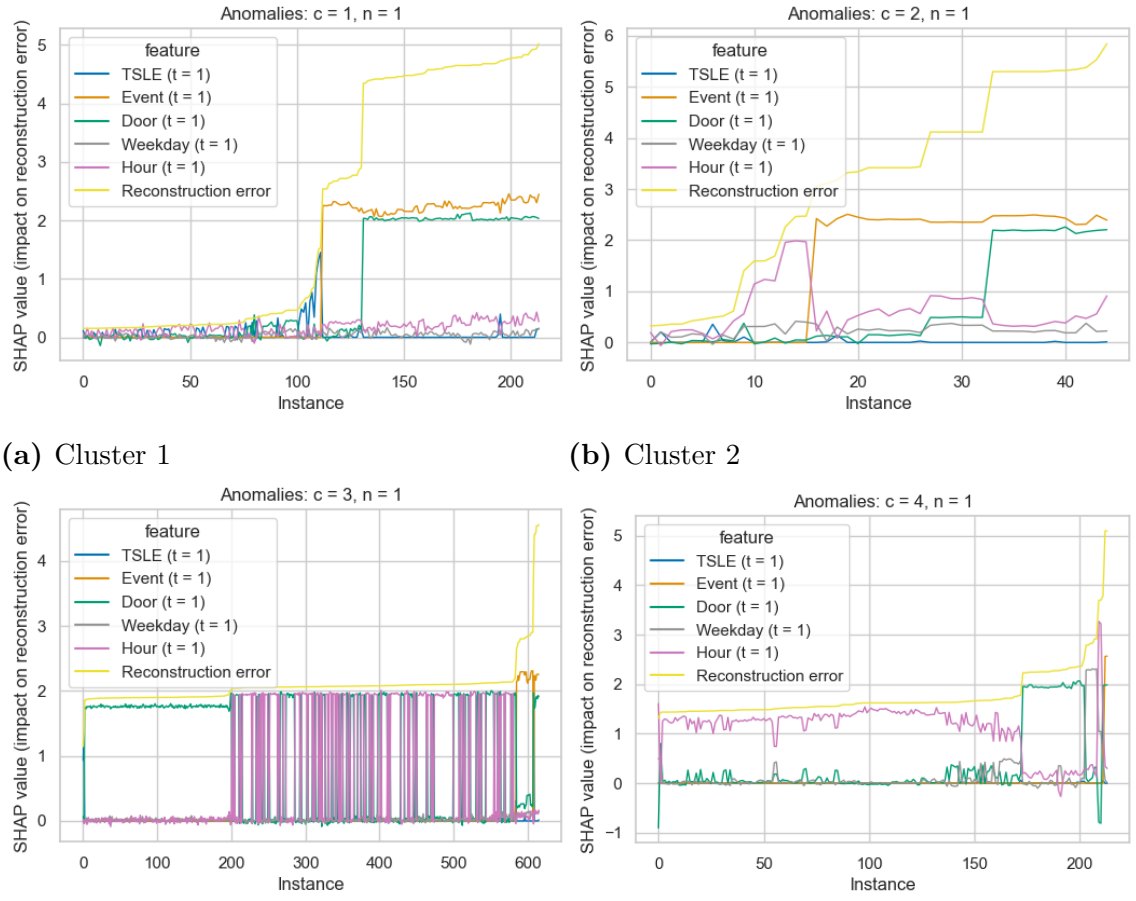


(a)



(b)

Figure 6.7: Distribution of SHAP values for all normal (a) and anomalous (b) instances for each feature in each cluster. This is based on predictions for $n = 1$. Each point corresponds to the average of 100 retrievals of the SHAP value for a particular instance.



(a) Cluster 1

(b) Cluster 2

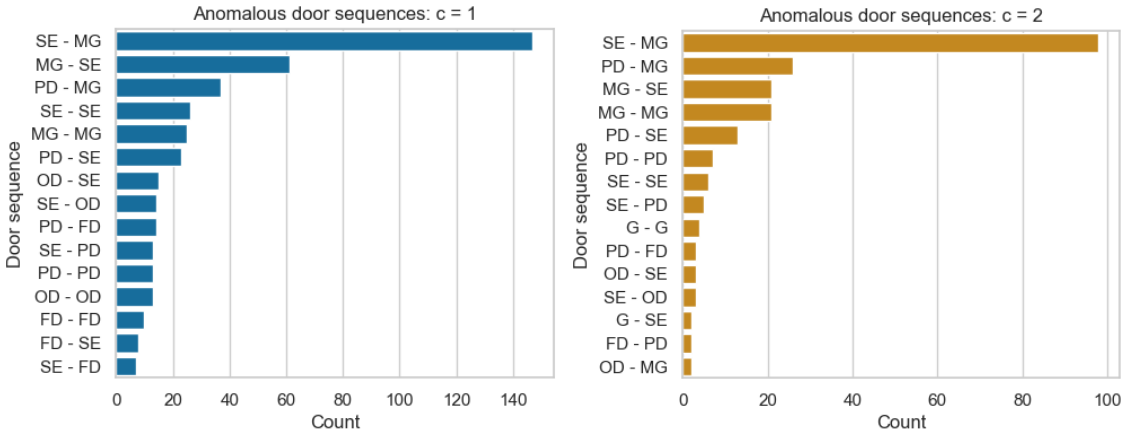
(c) Cluster 3

(d) Cluster 4

Figure 6.8: SHAP values plotted against the reconstruction error for each feature in each cluster. The SHAP value for each instance corresponds to the average of 100 retrievals.

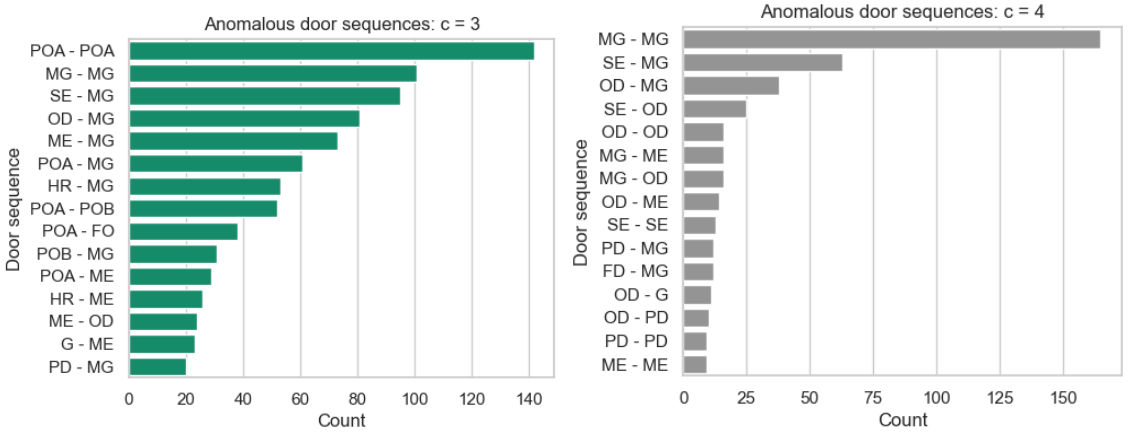
Cluster	Label	Event	Door	Hour	TSLE	Weekday	Timestamp
1	A1	UE2	FD	14	36,399	Tue	2020-03-03 14:50:13
	A2	UE4	SE	7	2,949,308	Thu	2020-04-23 07:55:04
	N1	UE1	SE	12	363,276	Mon	2020-03-23 12:45:36
	N2	UE1	MG	3	229,596	Mon	2020-07-06 03:35:44
2	A3	UE1	MG	1	166,895	Fri	2020-05-22 01:58:19
	A4	UE1	SE	23	67	Fri	2020-02-07 23:28:23
	N3	UE1	SE	21	180,125	Sun	2020-06-28 21:29:09
	N4	UE1	PD	11	11	Tue	2020-02-04 11:35:08
3	A5	UE1	OD	10	48,181,026	Thu	2020-01-23 10:48:21
	A6	UE1	ME	2	95	Fri	2020-06-26 02:57:57
	N5	UE1	FD	9	17,454	Wed	2020-02-12 09:57:25
	N6	UE1	ME	3	56,186	Wed	2020-05-27 03:36:49
4	A7	UE1	KC	4	14	Wed	2020-06-10 04:14:48
	A8	UE1	MG	4	71,833	Sun	2020-02-09 04:13:06
	N7	UE1	PD	22	9	Sun	2020-03-15 22:50:14
	N8	UE1	SE	4	144,444	Tue	2020-02-04 04:27:57

Table 6.1: Two anomalous and two normal instances for each cluster for $n = 1$.



(a) 33 unique sequences.

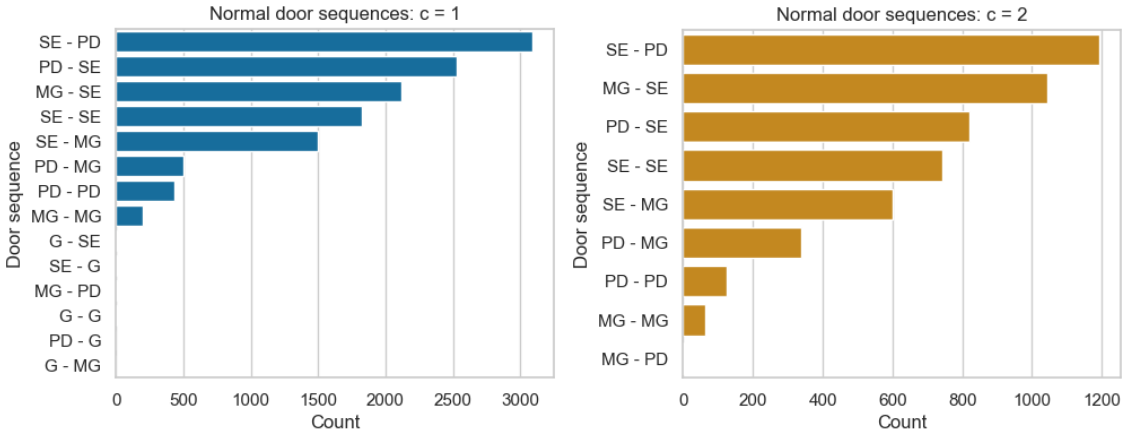
(b) 27 unique sequences.



(c) 108 unique sequences.

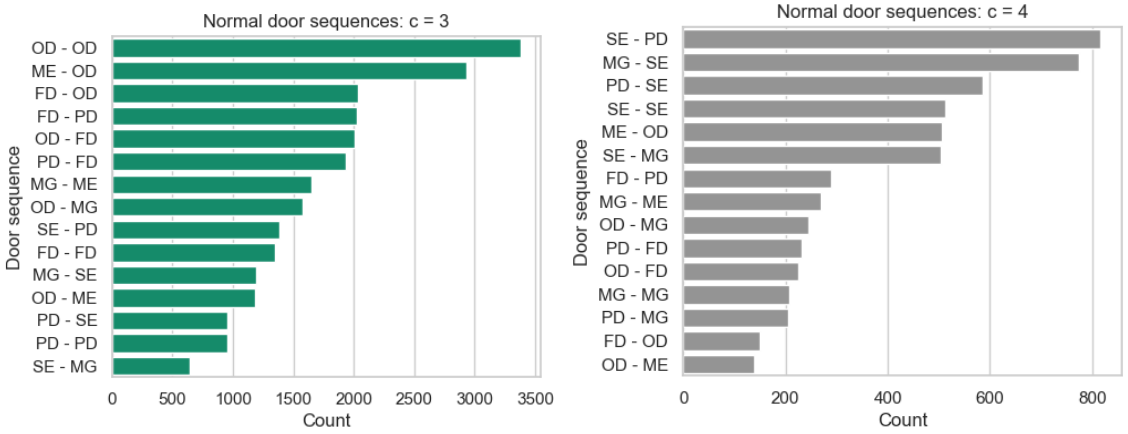
(d) 48 unique sequences.

Figure 6.9: The 15 most common sequences in each cluster for anomalous instances.



(a) 14 unique sequences.

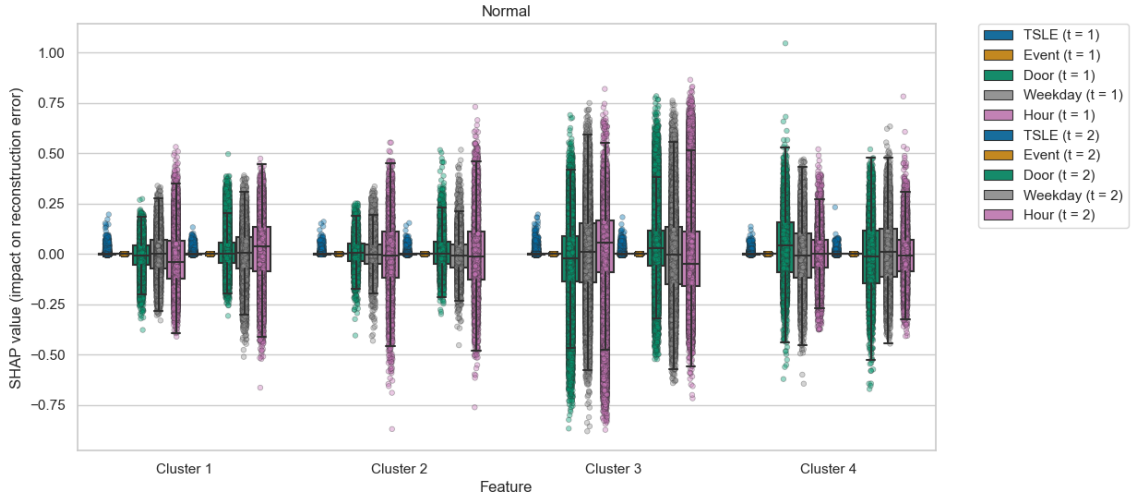
(b) 9 unique sequences.



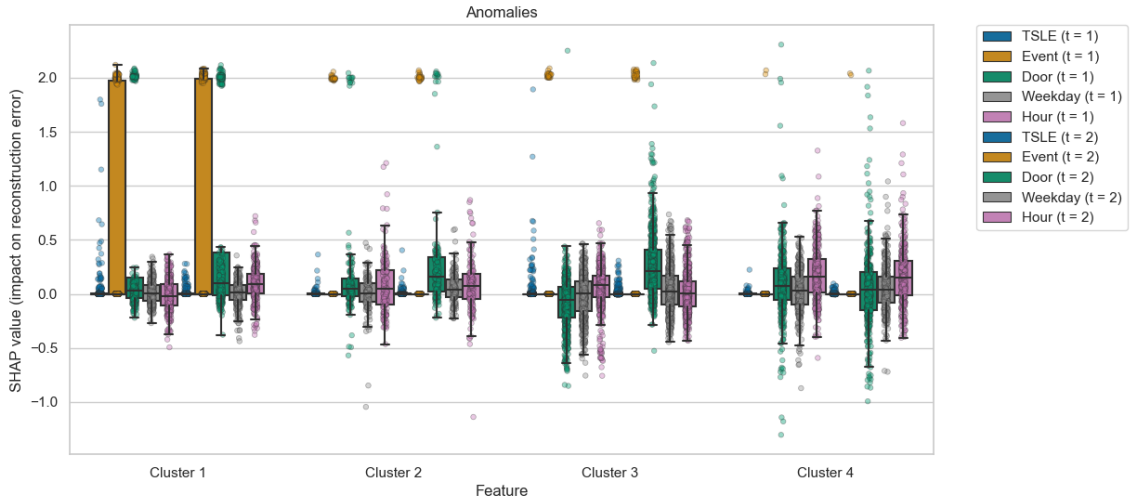
(c) 97 unique sequences.

(d) 37 unique sequences.

Figure 6.10: The 15 most common sequences in each cluster for normal instances.

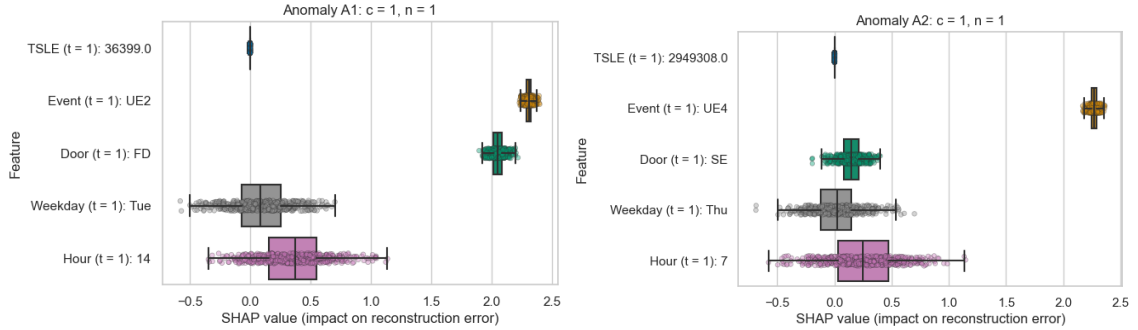


(a)

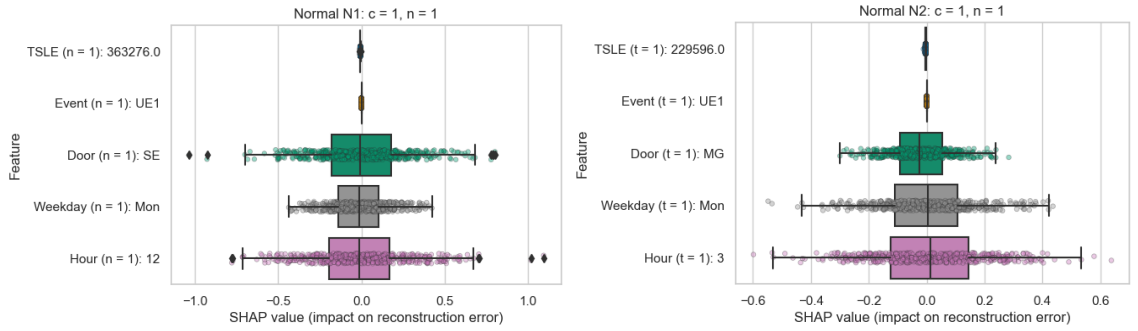


(b)

Figure 6.11: Distribution of SHAP values for all normal (a) and anomalous (b) instances for each feature in each cluster. This is based on predictions for $n = 1$. Each point corresponds to the average of 100 retrievals of the SHAP value for a particular instance.

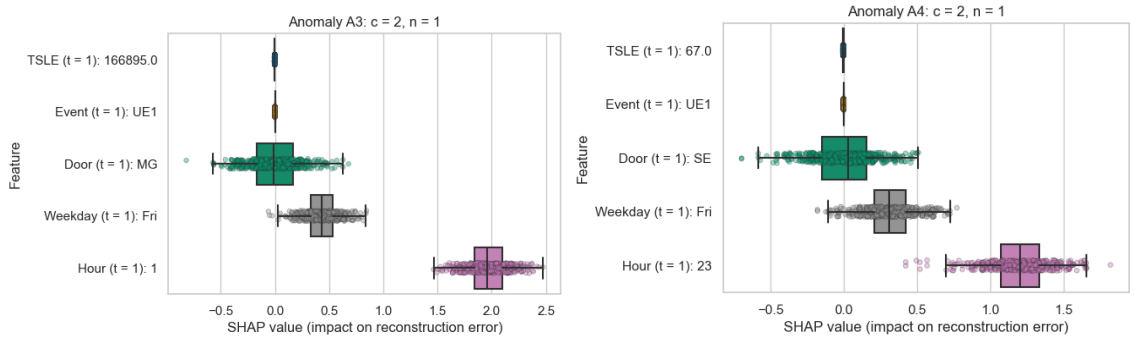


(a) (b)

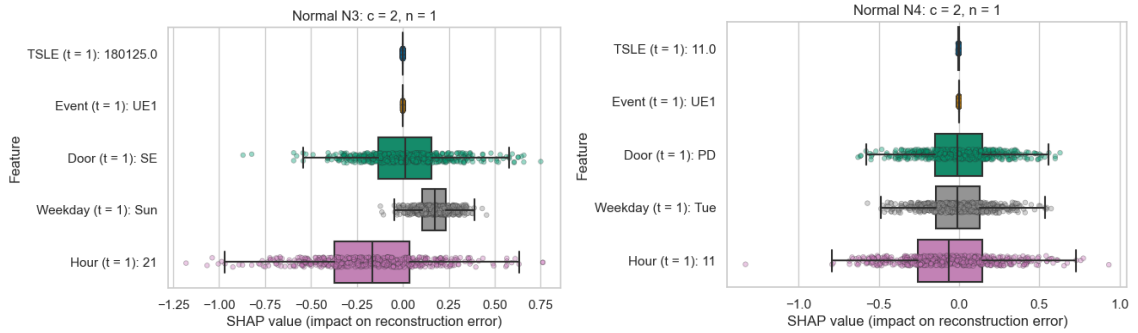


(c) (d)

Figure 6.12: Distribution of 500 retrievals of the SHAP values of each feature for two anomalous and two normal instances for $c = 1$ and $n = 1$.



(a) (b)



(c) (d)

Figure 6.13: Distribution of 500 retrievals of the SHAP values of each feature for two anomalous and two normal instances for $c = 2$ and $n = 1$.

Cluster	Label	Event	Door	Hour	TSLE	Weekday	Timestamp
1	A9	UE1	SE	11	374,654	Tue	2020-06-09 11:45:55
		UE1	SE	3	2,130,850	Sat	2020-07-04 03:40:05
	A10	UE1	SE	7	1,624,343	Tue	2020-03-31 07:59:37
		UE1	MG	8	1,996	Tue	2020-03-31 08:32:53
	N9	UE1	SE	12	89,583	Fri	2020-01-31 12:52:07
		UE1	PD	12	19	Fri	2020-01-31 12:52:26
	N10	UE1	PD	12	14	Mon	2020-02-24 12:46:16
		UE1	SE	12	85,015	Tue	2020-02-25 12:23:11
2	A11	UE1	PD	22	12	Fri	2020-02-21 22:44:04
		UE1	MG	8	122,102	Sun	2020-02-23 08:39:06
	A12	UE1	MG	11	172,646	Thu	2020-05-28 11:42:05
		UE1	MG	21	293,911	Sun	2020-05-31 21:20:36
	N11	UE1	SE	19	114	Wed	2020-04-15 19:42:16
		UE1	PD	19	14	Wed	2020-04-15 19:42:30
	N12	UE1	PD	20	21	Wed	2020-01-22 20:12:44
		UE1	MG	20	86648	Thu	2020-01-23 20:16:52
3	A13	UE1	SE	10	8,776	Fri	2020-08-21 10:49:23
		UE1	HR	11	1,382	Fri	2020-08-21 11:12:25
	A14	UE1	MG	11	75,161	Wed	2020-08-12 11:41:35
		UE1	OD	20	30,760	Wed	2020-08-12 20:14:15
	N13	UE1	MG	5	434,263	Mon	2020-04-27 05:08:14
		UE1	ME	5	127	Mon	2020-04-27 05:10:21
	N14	UE1	OD	6	17	Thu	2020-05-14 06:08:20
		UE1	POA	6	18	Thu	2020-05-14 06:08:38
4	A15	UE1	SE	4	87,162	Tue	2020-05-05 04:05:37
		UE1	OD	8	17,453	Tue	2020-05-05 08:56:30
	A16	UE1	MG	1	259,941	Thu	2020-07-09 01:22:15
		UE1	MG	0	81,696	Fri	2020-07-10 00:03:51
	N15	UE1	OD	19	12	Sun	2020-04-19 19:33:19
		UE1	FD	19	188	Sun	2020-04-19 19:36:27
	N16	UE1	FD	10	73	Wed	2020-01-29 10:58:50
		UE1	PD	10	13	Wed	2020-01-29 10:59:03

Table 6.2: Two anomalous and two normal instances for each cluster for $n = 2$. Note that in this case an instance corresponds to a sequence of two events.

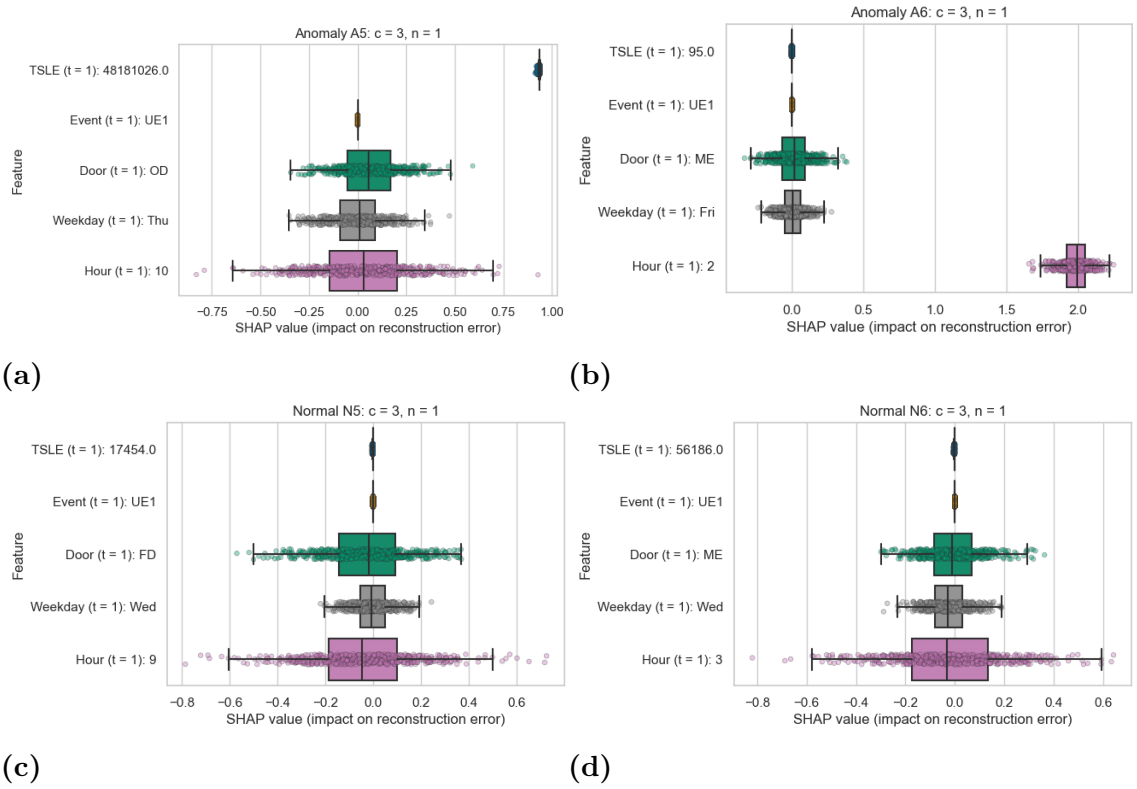


Figure 6.14: Distribution of 500 retrievals of the SHAP values of each feature for two anomalous and two normal instances for $c = 3$ and $n = 1$.

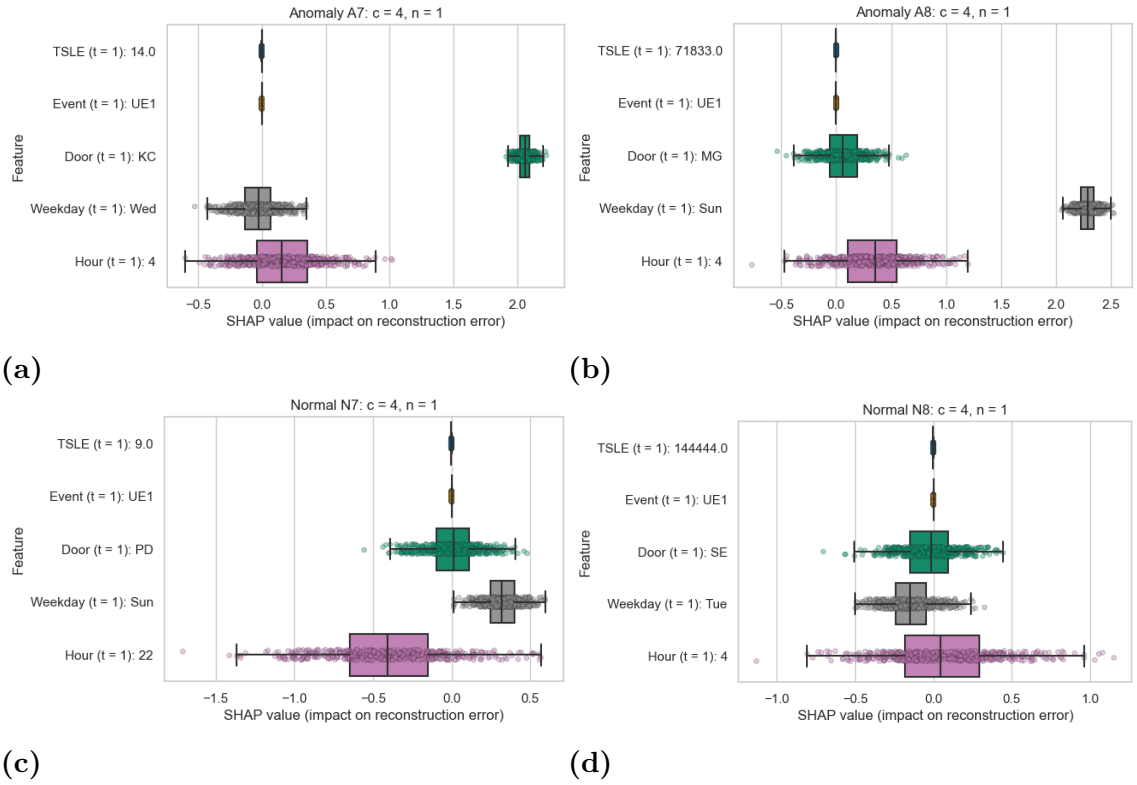
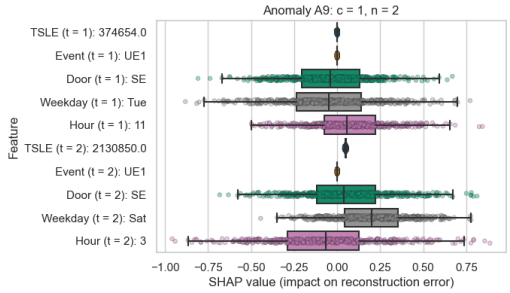
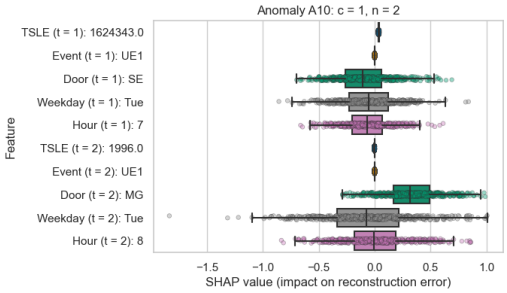


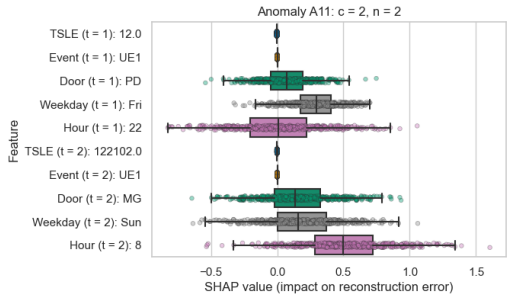
Figure 6.15: Distribution of 500 retrievals of the SHAP values of each feature for two anomalous and two normal instances for $c = 4$ and $n = 1$.



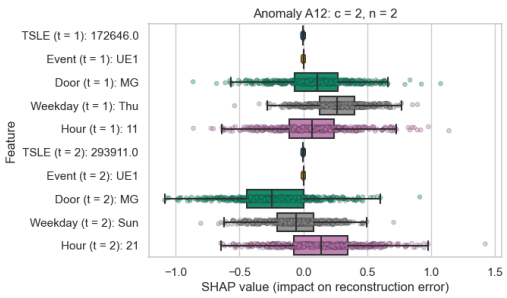
(a)



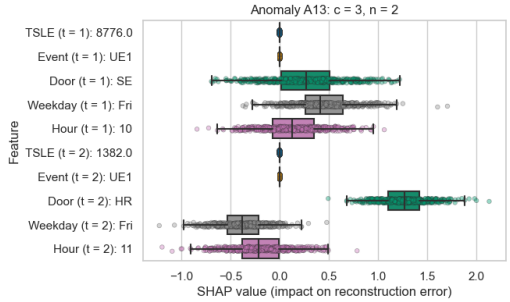
(b)



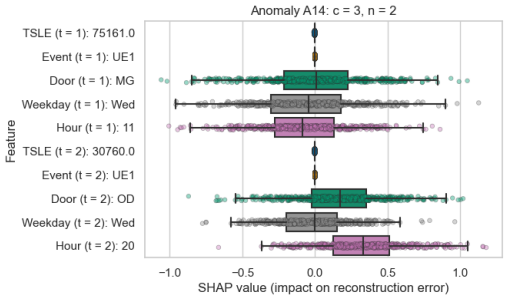
(c)



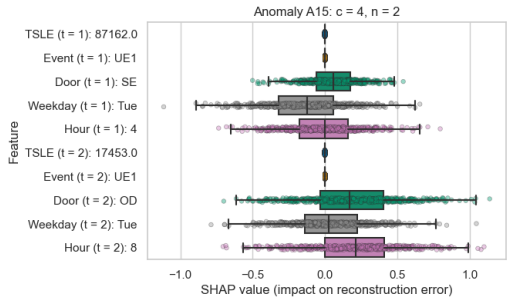
(d)



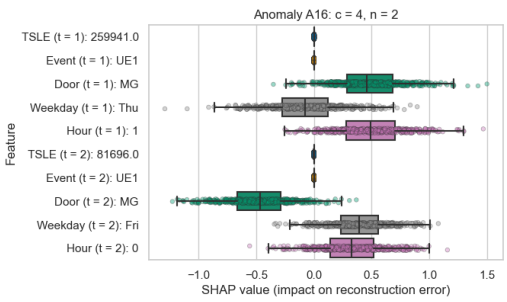
(e)



(f)



(g)



(h)

Figure 6.16: Distribution of 500 retrievals of the SHAP values of each feature for two anomalous instances for all clusters for $n = 2$.

7
Discussion and Conclusion

This chapter will conclude the thesis by discussing the results. Sections 7.1-7.4 follow the same structure as the results presented in the previous chapter and discusses each of these. Section 7.5 describes a few ethical considerations. Section 7.6 lists various conclusions made based on the results and suggests possible future work that can further improve the results. Finally, Section 7.7 describes the final conclusion of the thesis.

7.1 Training Process

From the plots in Figure 6.1 it is clear that all eight datasets contain common patterns that each of the eight models are able to learn. Furthermore, the validation set (last 10% of the training set) seem to follow somewhat similar patterns since the loss is monotonically decreasing at each epoch for the validation sets in all cases. In short, no overfitting or underfitting occurred during the training phase.

7.2 Training Thresholds

From the plots in Figures 6.2 and 6.3 one can see that most instances are grouped below the red line ($T_{c,n}$), which shows that most events are considered normal. The anomalous instances, on the other hand, are much fewer and are scattered above the red line. Deciding the threshold is a balance between minimizing false positives and false negatives. In certain cases it could be argued that the threshold is perhaps a bit too low (e.g. plot (a) from Figure 6.2).

Furthermore, it should be noted that for the plots (e) and (g) from Figure 6.2, one can see that there is a cluster of instances formed above the threshold line. These instances may correspond to events that are rare compared to remaining instances, but could still correspond to somewhat normal behavior. A solution to this could be to decide the thresholds $T_{c,n}$ by using multipliers in Equation 5.7 that are specifically suited for the specific cluster and value of n. For practical usage, this method should likely be applied. The method used in this project only leads to more false positives (i.e., anomalies that are in reality not anomalous), which does not largely impact the rest of the evaluation process. The reason is that the false negatives should not be impacted, and the true anomalies are most important.

7.3 Global Evaluation

This section presents the global evaluation of the point anomalies ($n = 1$) and collective anomalies ($n = 2$).

7.3.1 Point Anomalies

The plots in Figure 6.4 should be compared to the overall door access behavior shown in Figure 5.2. Given this, it is clear that the anomalous door accesses shown in plot (b) deviate much more from the most common behavior. For example, users from cluster 1 and 2 are accessing the office related doors, which they do not have permission to use. This is again because the test data includes user events of type (`U2`, `U3` and `U4`). Furthermore, some of the really uncommon doors like `FO` and `G` are essentially only considered anomalous whenever they appear. Whether this makes sense or not depends on the context. Since it is a door that is sometimes used, it is likely not an anomaly corresponding to harmful behavior. However, since it is so uncommon that the models believe that it is anomalous, perhaps it may as well be reported to administrators. In other words, it is uncommon enough to not lead to a lot of unnecessary manual work. Alternatively, increasing the threshold slightly would likely cause these instances to become normal as well, with the risk of getting more false negatives.

The plots in Figure 6.5 and 6.6 should be compared to the overall access behavior shown in Figures 5.3 and Figures 5.4, respectively. For both of these cases one can draw similar conclusions as for the door accesses. In other words, the anomalous instances show a very different access pattern compared to the overall access behavior.

Next, from Figure 6.7 it becomes clear that the SHAP values of anomalous instances are generally larger. In fact, the mean SHAP value across all instances is above zero for most features for the anomalous instances. For normal instances the mean is around zero. Furthermore, it is clear that the features `Event`, `Door` and `Hour` have the largest impact on the anomalies.

Similar conclusions can be drawn for Figure 6.8. However, this figure also reveals that the features `Event` and `Door` seem to be responsible for the largest reconstruction errors. This is especially true for clusters 1 and 2. In both of the plots (a) and (b) there is an initial jump in the reconstruction error, credited to the feature `Event`. Then there is another jump slightly after, which corresponds with an increase of the feature `Door`. This occurs because `Event` can be anomalous by itself if a user attempts to open a common door within the cluster, but the door not opening. When both `Event` and `Door` have high SHAP values, it corresponds to a user attempting to open a disallowed door, that is also uncommon in general within the cluster. There will be examples of this in Section 7.4.

From the same figure one can also identify the feature `Hour` having a large impact, as well as `Weekday` and `TSLE` occasionally increasing. Finally, the figure reveals that the chosen threshold for cluster 1 (and potentially cluster 2) should have been higher.

The reason being that the reconstruction error is quite low for the first 80 or so instances.

7.3.2 Collective Anomalies

From Figures 6.9 and 6.10 one can see that the sequences that correspond to anomalous instances generally involve doors that are uncommonly accessed. This includes sequences involving `MG` for all clusters or some of the office related doors for $c = 3$. However, it is also clear that certain common sequences such as {`SE - PD`} and {`ME - OD`}, indicating that there are sequential anomalies caused by features other than the doors being accessed in individual time steps.

From Figure 6.11 one can again see that the SHAP values for anomalies are generally larger and more spread, compared to normal instances. The SHAP values for normal instances again average around zero, similar to the overall SHAP values for point anomalies. The one thing that differs for the collective anomalies is that there are not as many individual instances that deviate significantly. However, the reconstruction errors are around the same as for the point anomalies (see Figure 6.3). Furthermore, the SHAP values for all features for a given instance should sum to the reconstruction error of the same instance [4]. This indicates that the SHAP values are more frequently spread among multiple features for collective anomalies. The SHAP values for point anomalies are instead be higher for single features. This will be further confirmed in the next section. This does make sense considering that the sequential instances have twice as many features that impact the reconstruction error.

7.4 Local Evaluation

This section discusses the eight chosen instances for point anomalies ($n = 1$) and collective anomalies ($n = 2$).

7.4.1 Point Anomalies

Each instances is listed and discussed below. Details of each instance can be found in Table 6.1. Also, the SHAP value for each feature and instance is shown in Figures 6.12-6.15. Note that box plots were used to visualize the spread of the SHAP values. However, the mean SHAP value, which is indicated in each box plot, is used when deciding the significance of a particular feature.

- **Cluster 1**.
  - **A1**. This anomaly corresponds to the event `UE2`, which means a user attempted to open an invalid door for the given access category. The SHAP value for the feature `Event` is consequently high for this anomaly. Additionally, the SHAP value for the door `FD` is high. This means that a

user attempted to open the door FD which is invalid, and therefore uncommon within cluster 1. The hour (14) seems to have a slight impact as well, which makes sense since accesses at this hour is relatively uncommon for cluster 1 (see Figure 5.4).

  - **A2**. This anomaly is similar to A1 in that the event (UE4) corresponds to a high SHAP value. However, in this case the door has a low SHAP value. This is because UE4 corresponds to a door being temporarily inhibited. This shows that doors that are very common within a cluster (e.g. door SE for cluster 1 in this case), can still be temporarily disallowed. Thus, the SHAP values correctly indicate that it is anomalous only because of the event and not also the door in this case.

  - **N1**. This instance shows an example of when the door SE being accessed is considered normal. The box plots indicate that the mean SHAP values are around zero, indicating that none of them are contributing to it being anomalous. Opening SE at noon on a Monday is therefore considered normal for users in cluster 1, as expected.

  - **N2**. This instance shows that opening the main gate MG on a Monday at roughly 3 am is normal behavior for users in cluster 1. Again, the SHAP values average around zero for all features. This will become a common theme for instances that correspond to normal behavior.

- **Cluster 2**.

  - **A3**. This anomaly indicates that accessing MG at 1 am on a Friday is abnormal. Furthermore, the SHAP values indicate that it is mainly due to the time (1 am), and slightly because of the weekday. This is logical considering the overall behavior within cluster 2. Note that there are many similar instances where MG is accessed during the night shift. This indicates that the main gate is likely remained open during the day, such that users do not have to open it manually using their access cards.

  - **A4**. This anomaly is similar to A3, except that it is accessing the door SE. In this case, despite the weekday being the same and the hour being equally uncommon, the SHAP value for the hour is quite a bit lower. While still being an anomaly, it is closer to being normal which is likely due to the door SE being more common than MG.

  - **N3**. This instance indicates that accessing SE on a Sunday at 9 pm is common behavior. The SHAP values show that the weekday (Sunday) does contribute slightly to it being an anomaly. However, the hour (9 pm) has a negative SHAP value which offsets the instance enough to not be considered anomalous. In other words, this shows that if the hour would have been something more uncommon then it would likely have been anomalous.

  - **N4**. Finally, accessing PD on a Tuesday at 11 am is expectedly common behavior for cluster 2. This is also indicated by the SHAP values.

- **Cluster 3**.

  - **A5**. The SHAP values for this instance indicate that accessing OD on a Thursday at 10 am is normal for cluster 3. However, the instance is classified as anomalous due to the feature TSLE, which is further indicated by the SHAP value. The value of TSLE for this instance is 481,810,26 seconds, which corresponds to 557 days. In other words, this is the first event recorded from this user ID in 557 days.

  - **A6**. This instance shows that accessing the main entrance (ME) is considered common based on the SHAP values. However, accessing ME on a Friday at 2 am is uncommon for cluster 3, thus making this instance anomalous.

  - **N5**. Opening FD on a Wednesday at 9 am is expectedly common behavior for cluster 3.

  - **N6**. Opening ME on a Wednesday at 3 am is expectedly common behavior for cluster 3.

- **Cluster 4**.

  - **A7**. As indicated by Figure 5.2, the key cabinet (KC) is very uncommon for cluster 4. Instance A7 is therefore anomalous mainly because of this, as is also indicated by the SHAP values.

  - **A8**. The SHAP values for this instance show that it is mostly anomalous due to the weekday. However, Figure 6.5 showed that this cluster does not consider all instances on Sundays to be anomalous. Instead, it is the fact that it is opening MG at 4 am on a Sunday that makes it anomalous. Note that neither MG or the hour 4 is that uncommon for cluster 4. This indicates that it is an unusual combination of three relatively usual features that caused the anomaly. Note further that this conclusion could not be made by simply inspecting the SHAP values. In other words, comparing the SHAP values to common behavior within the clusters is important in order to retrieve the full picture.

  - **N7**. Here is an example of an instance that corresponds to normal behavior with an access on a Sunday. In this case it is accessing PD at 10 pm which means that this combination of features is more common than what was shown in anomaly A8.

  - **N8**. Finally, accessing `SE` on a Tuesday at 4 am is normal behavior for cluster 4.

7.4.2 Collective Anomalies

All instances are listed and discussed below. Details of each instance can be found in Table 6.2. Also, the SHAP values for all the anomalous instances are shown in Figure 6.16. The SHAP values for normal instances are purposely left out in this case as they all tend to average around zero, meaning that similar conclusions will be drawn as for the normal point instances.

All instances presented in this section correspond to sequences of events, where each event in each sequence is considered normal by the models for $n = 1$. This means that none of the anomalous sequences below are considered anomalous because of one of the events in the sequence being anomalous. Instead, they are anomalous only because the particular sequence itself is abnormal. This was done to ensure that this section provides something new compared to the previous section where the point anomalies were presented.

- **Cluster 1**.

  - **A9**. This instance is anomalous because of the time between the two events. The SHAP value of `TSLE` for $t = 2$ slightly indicates this. This feature shows that there was 2,130,850 seconds (or roughly 24 days) between the two events. This of course means that there will be no relation between the two events, thus making it anomalous. An interesting note here is that the sequence is considered anomalous both because of the fact that there is no relation between the events (due to the time between them), as well as because of the time between them (due to the feature TSLE being included).

  - **A10**. This instance consists of the door access sequence {`SE - MG`}. This sequence was the most common sequence among the anomalous instances for cluster 1, which indicates that it should be anomalous due to the doors. The SHAP values confirm that this is the case. The SHAP value of `Door` at $t = 1$ is negative, showing that the door `SE` being accessed at $t = 1$ is common, and not the cause of the anomaly. However, the SHAP value of `Door` at $t = 2$ is the highest, showing that `MG` being accessed after `SE` is uncommon, which makes sense since the main gate is generally accessed before the staff entrance.

  - **N9**. This sequence corresponds to {`SE - PD`}, which is the most common sequence for this cluster (see Figure 6.10). Furthermore, the remaining features correspond to normal values, thus logically making it a normal sequence.

  - **N10**. This sequence corresponds to {PD - SE}, which is the second most common sequence (see Figure 6.10) for this cluster. This shows that the model has found patterns for sequences that do not take place in the same day. In this case, it corresponds to the last event of the day, followed by the first event the next day. The time between these two events (indicated by TSLE at $t = 2$) is 85,015 seconds which is roughly 24 hours.

- **Cluster 2**.

  - **A11**. This instance corresponds to {PD - MG}, which is another example of a sequence of door accesses that take place on different days. The instance N12 (see discussion below) shows the same door sequence but instead when the model considered it normal. In this case it seems that it was considered anomalous due to the weekdays at both time steps ({Fri - Sun}, as well as the time of the second time step (8 am).

  - **A12**. This anomaly shows two consecutive accesses of MG, on two separate days. What is interesting about this anomaly is that the individual accesses of MG are completely normal for cluster 2, while them appearing in sequence is not normal. In other words, a user seems to only be accessing the main gate and nothing else on two separate days. It is not clear whether this implies harmful behavior, but it is nonetheless worth investigating and it would not have been possible to detect if only point anomalies were being considered.

  - **N11**. Again, the sequence {SE - PD} is also very common for cluster 2.

  - **N12**. As already mentioned, this instance corresponds to {PD - MG}. Unlike A11, this instance is considered normal since remaining features are more common compared to A11.

- **Cluster 3**.

  - **A13**. This instance corresponds to the door sequence {SE - HR}. Note again that both of the individual events in this sequence is considered normal by the point anomaly model for cluster 3. In other words, despite the door HR being uncommon, it is not anomalous by itself (i.e., not a point anomaly). However, this instance makes it clear that HR being accessed after SE is uncommon. The SHAP values further indicates that it is mostly due to HR appearing at $t = 2$. The anomaly makes sense considering that HR is an office related door, and is generally accessed after OD or one of the other office doors.

  - **A14**. This instance corresponds to the door sequence {MG - OD}. This is a very unusual sequence of door accesses. Generally, one of the building

entrances are accessed after MG (i.e., SE or ME). The SHAP values indicate slightly that it is due to OD appearing at $t = 2$. Furthermore, the time of the access at $t = 2$ (8 pm) is contributing to the anomaly as well. This anomaly is a potential example of a broken sequence, which is an already mentioned issue (see Chapter 4) with the access logs. In this case it is likely that the true sequence taken by the user was {MG - ME - OD}. However, for some reason the access to ME was no logged, leaving only {MG - OD}, and is therefore falsely assumed to be anomalous. This is further discussed later in this chapter.

- **N13**. This instance corresponds to the door sequence {MG - ME}. This is one of the seventh most common sequence among normal sequences for cluster 3 (see Figure 6.10 (c)).

- **N14**. This instance corresponds to the door sequence {OD - POA}. This sequence expectedly shows how accessing the office door OD followed by one of the office related doors (POA) is considered normal.

- **Cluster 4**.

  - **A15**. This instance corresponds to the door sequence {SE - OD}. This is another example of a broken sequence that is interpreted as an anomalous instance. The reason is that there is no path directly from SE to OD. As indicated by Figure 4.7 in Chapter 4, there is a common sequence of length four that achieves this path, which is {SE - PD - FD - OD}. In other words, the subsequence {PD - FD} seems to be missing.

  - **A16**. This instance corresponds to the door sequence {MG - MG}. According to TSLE for $t = 2$ there are roughly 24 hours between the two accesses. This indicates that the user only accessed MG on the first day in the sequence (Thursday). This means that it may again be a case of a broken sequence. As was shown in Figure 6.9, this sequence is the most common among the anomalous sequences for cluster 4. However, Figure 6.10 also showed that there are cases when this sequence is considered normal, which shows that these broken sequences may be common enough to be considered normal. From the SHAP values one can see that it is positive for the door at $t = 1$, but negative for the door at $t = 2$. This may indicate that the door MG is generally more common at $t = 2$, compared to $t = 1$ for this particular sequence of features.

  - **N15**. This instance corresponds to the door sequence {OD - FD}. This sequence together with the next instance N16 illustrates that the sequence {OD - FD - PD} is a common sequence that allows movement between the production area and the office.

  - **N16**. This instance corresponds to the door sequence {FD - PD}. See

note for `N15` above for the discussion.

7.5 Ethical Considerations and Sustainability

There are some ethical considerations concerning this project. The most important aspect is in regard to machine learning. If the anomaly detection system identifies that a specific user caused an anomaly, the trustworthiness of this assessment must be further evaluated. Training a machine learning model to be perfect at detecting anomalies is very difficult, if not impossible, and should therefore not be blindly trusted. Any action taken must not be based purely on trusting the output as this could potentially be a false positive. Specifically, if the model incorrectly considers a normal instance to be anomalous, this must not result in inappropriate actions being taken. The developed framework should therefore be seen as a tool to aid security administrators in making assessments easier.

Training and using a machine learning model for the purpose of anomaly detection has an energy cost that is substantially higher than simply inspecting access logs manually. However, as mentioned before, manually inspecting access logs is not a feasible method in preventing attacks. As such, using a machine learning approach is preferable despite the energy cost. Being able to prevent attacks from happening is of course more sustainable than, for instance, having to replace equipment and material due to theft and vandalism.

7.6 Conclusions and Future Work

This section presents the conclusions and possible future work. A number of points are discussed separately in the list below.

- **Reconstruction error thresholds.** Deciding the thresholds $T_{c,n}$ is a balance between minimizing false negatives and false positives. False negatives (i.e., predicting an instance to be normal when it is in fact anomalous) is much more severe in this case. The reason is that more false negatives mean that harmful behavior may go unnoticed. The amount of false negatives is minimized by having a lower threshold. A threshold that is too low, however, will lead to a larger number of false positives which increases the amount of unnecessary manual labor required by administrators.

  The current way of analyzing the access logs would be to have administrators check every single event manually. Due to the size of the logs this instead ends up not happening at all, which means that the maximum number of false negatives is achieved. The system proposed in this thesis attempts to minimize the amount of false negatives to ensure that most malicious behaviour is correctly identified. Furthermore, the reconstruction error defines the level of rarity of a particular event in the access logs which can be used as a subjective

ranking of the level of harm that they may cause. While this assumes that rarity implies harmful behaviour, one can at least use this as a way to prioritize which events to investigate further.

- **SHAP values.** For $n = 1$ the SHAP values seem to make sense for all the 16 instances shown. In other words, the SHAP values have been large for features with values that correspond to uncommon behavior for respective cluster, for every anomalous instance. For $n = 2$ the SHAP values are in certain cases not very large for one or two features. Instead, the SHAP values are smaller and spread among multiple features. This indicates that multiple features contributed to the anomalous reconstruction error. This makes sense considering that each instance at each time step is considered normal by the point anomaly models. Because of this, the anomalies for $n = 2$ are generally caused by an unusual combination of features, and not due to any individual features being the only contributors to the anomaly.

  As explained in Section 5.4, SHAP values were possible to calculate because of an additional layer added to the original LSTM autoencoder models. This extra layer outputs the reconstruction error directly. A different approach, which attempts to calculate SHAP values in a more thorough fashion, was outlined by Antwarg *et al.* [5]. This approach instead outputs a vector containing the reconstruction errors for each individual feature. This makes it possible to analyze the effects that each input feature has on the reconstruction error for each individual feature. The approach used in this project, only looks at the overall impact of each input feature on the full reconstruction error. Note that the full reconstruction error is essentially the sum of the reconstruction errors of each individual feature. Investigating this alternative method of retrieving SHAP values could be interesting as future work.

- **Usefulness of features.** The features used for the machine learning models was `TSLE`, `Weekday`, `Hour`, `Door` and `Event`. Out of these, `Weekday`, `Hour` and `Door` were used to find subgroups of users with similar behavior. This was of course an arbitrary choice which can likely be improved. For example, one could cluster groups of users based on common combinations of these features and not just looking at each feature separately. For example, within cluster 3 there may be additional subgroups of users that tend to open similar doors at similar hours, with users from each subgroup still having similar overall access counts. That sort of relationship is not captured in this project since only the raw access counts were considered. The clustering may also be done based on similar sequences taken, and not just individual doors.

  For the machine learning models it seems that all features have an impact on which instances are considered anomalous. The only questionable feature would be `TSLE`, partly because of instances like `A5` above. This instance is anomalous because it was the first event for this particular user ID in 557 days. First of all, the reason for such events is worth speculating on. One theory is

that the user stopped working and then came back after a year and a half. A different theory is that a new person was employed but assigned the same user ID as the old user in the PAC system. This may be one of the causes for why the access category of certain user ID's changed at some point in the access logs. For these cases it is not an issue, but if this occurs without a change of access category it is completely unknown based purely on the access logs. This means that a change of behavior from one user ID may go unnoticed. The problem is naturally avoided by assigning unique user ID's to every new user, and excluding user ID's of previously employed users in the pool of available ID's.

Now, the intended idea of `TSLE` was originally to detect unusual times between accesses in the order of seconds, minutes or at most hours. This would, for example, allow catching anomalies where a user accesses two doors that are far apart in an abnormally short time interval. However, the extremely large values of `TSLE`, as observed for instance `A5`, make it useless when detecting abnormalities in the order of seconds, minutes or hours. The reason being that a hundred seconds will have no impact on the reconstruction error when abnormal values of `TSLE` is in the order of many millions of seconds. The reason for why such large values of `TSLE` show up is because of how the value of this feature was assigned to each event. It was assigned by simply taking the time since the last event, individually for all events of each user. An alternative approach could be to assign a special value (perhaps zero) whenever the previous event is from a different working shift. This means that the extremely large values will be eliminated. This could be interesting to try as future work. However, doing this means that anomalies such as `A5` would no longer be detected, which may or may not be negative depending on the context.

- **Broken sequences.** The issue of broken sequences was already discussed in Chapter 4. When looking at anomalous sequences in Section 7.4.2, it became clear that some of them were anomalous due to broken sequences. This further strengthens the potential issue caused by these broken sequences. Some broken sequences may be common, in which case the machine learning models will assume that it is normal behavior. This means that the model would not catch this sequence as an anomaly even in cases where it might be caused by potential harmful behavior. On the other hand, when certain broken sequences are not common, they will be assumed to be anomalous by the models. This will not lead to false negatives, but instead cause a lot of extra unnecessary manual work by administrators. However, the broken sequences that are reported by the models as being anomalous could (whether it is harmful or not) give clues to administrators that an access path into a section of the building needs to be fixed (if security in this part of the building is especially important). Additionally, it may also be used to identify users that tend to frequently tailgate. Furthermore, one could assume that harmful behavior occurs when the user is alone, requiring access to all doors using their own access card. This implies that there would not be a broken sequence whenever an anomaly

corresponds to harmful behavior.

The above discussion of broken sequences is not so concrete mainly because it is not so clear how big of an issue it is. Figuring this out in a more concrete fashion would take a lot more analysis work. Both analysis of existing data as well as evaluations of the anomaly detection system after deployment by domain experts.

- **Dataset shift.** A clear limitation to the approach used is that user behavior can change over time. This means that a particular user is assigned to cluster c based on their training data activity, but then follows completely different behavior upon employment for various reasons. Furthermore, the behavior of a user could change drastically during the span of the training data. This could lead to a not so robust division of users based on their behavior. It may therefore be of interest to find clusters as well as train the LSTM autoencoders only on the latest data, such as the past six months. This approach maximizes the chances that the anomaly detection is done based only on the most relevant patterns in the data. However, doing this means that a great majority of the data must be dropped which means that certain (older) patterns will be missed by the models. Whether this is problematic depends entirely on how relevant the older patterns are and requires further evaluation to decide.

- **Machine learning benefits.** There are several benefits to using machine learning for the purposes of this project. One benefit is that the machine learning models are able to learn what is normal with respect to each of the features. The only feature where this is perhaps not so relevant is `Event`. The reason being that one could easily track when any of the events `UE2`, `UE3` or `UE3` occurs in the access logs, without machine learning. Furthermore, the models are capable of learning unusual combinations of features where the individual value of each feature is normal. Additionally, the use of LSTM autoencoders is a very suitable and powerful technique for the given data as explained in Section 5.2.

- **Practical usefulness.** The results presented in Chapter 6 were discussed with a domain expert from Bodforss Consulting AB. The conclusion from these discussions is that the suggested anomaly detection framework could be useful in practice since it is clearly reducing the amount of manual labor needed. To which degree it is useful is hard to speculate on before it has been deployed and tested for some time.

7.7 Final Conclusion

Physical access control systems are used to restrict access to various parts of a company building for security reasons. These systems tend to store access logs that contain a history of all accesses made by users in the system using their access

card. These access logs can however end up becoming large and hard to interpret. As such, security assessment becomes impractical, leading to the access logs rarely being examined. This thesis showed how analysis of these access logs can be used to detect subgroups of users that follow similar behavior. Furthermore, a significantly smaller subset of the original access logs that correspond to unusual behavior (with respect to overall behavior) could be identified using LSTM autoencoders. To be precise, the test set contains 60,430 events where 1,169 point anomalies and 1,496 collective anomalies were identified across all subgroups (based on the chosen thresholds $T_{c,n}$). This corresponds to a 95.6% decrease in the number of events that need to be manually inspected. Also, explanations for why the model considered each anomaly to be anomalous was generated using SHAP values. This means that a better understanding of what sort of anomalies exist in the system can be established. Understanding what typical anomalies look like can be used by administrators to proactively prevent them from occurring in the future. Additionally, it could be used to further improve the anomaly detection system itself (e.g. by being the basis of the labeling process for future training data).

There were two main issues that complicated the development of the system. The first is due to the inconsistency of the logs. The second is about the subjective evaluation caused by not knowing what an anomaly should look like in practice. The reason for the inconsistency is mainly caused by misusage (whether it is intentional or not) of the PAC system by the users. Full consistency is achieved if every single access made by every user is properly logged and if the state of the PAC system is the same throughout the span of the logs. The state refers to what behavior is possible within the building (defined by existing doors and so on). Furthermore, logging accesses in both directions of every door (and not just upon entry) would further improve the anomaly detection capabilities as this allows the models to properly understand when a user leaves the building. Ensuring that all (or most) accesses are logged is achieved by convincing users to always use their access card (i.e. avoid tailgating). Ensuring that the state of the PAC system is the same corresponds to mitigating a shift of the dataset, which could be solved by only utilizing the most recent part of the logs. Finally, solving the issue of subjective evaluation comes down to understanding beforehand which threats are possible such that one can manually label the access logs. This would require more domain specific analysis of the particular building where the anomaly detection system is to be deployed. However, it is clear that solving one (or both) of these issues is likely impractical in most cases. It should therefore be noted that the system developed in this thesis is still a useful start despite the mentioned issues not being minimized or fully solved.

References

[1] Verizon. *2020 Data Breach Investigations Report*. 2020. [Online]. URL: https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf. Accessed: 2021-03-01.

[2] Veriato. *Insider Threat Report 2018*. 2018. [Online]. URL: https://www.veriato.com/resources/whitepapers/insider-threat-report-2018. Accessed: 2021-04-24.

[3] Jinwon An and Sungzoon Cho. "Variational Autoencoder based Anomaly Detection using Reconstruction Probability". In: *Special Lecture on IE* 2.1 (2015), pp. 1–18.

[4] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Red Hook, NY, USA: Curran Associates, Inc., 2017, pp. 4765–4774. URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[5] Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. "Explaining Anomalies Detected by Autoencoders Using SHAP". In: *CoRR* abs/1903.02407 (2020). arXiv: 1903.02407v2. URL: http://arxiv.org/abs/1903.02407v2.

[6] Gerald L. Kovacich and Edward P. Halibozek. "Chapter 21 - Physical Security". In: *Effective Physical Security (Fourth Edition)*. Ed. by Lawrence J. Fennelly. Butterworth-Heinemann, 2013, pp. 339–353. DOI: 10.1016/B978-0-12-415892-4.00021-3.

[7] Francis Y. Edgeworth. "XLI. On discordant observations". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 23.143 (1887), pp. 364–375. DOI: 10.1080/14786448708628471.

[8] Dorothy E. Denning. "An Intrusion-Detection Model". In: *IEEE Transactions on Software Engineering* SE-13.2 (1987), pp. 222–232. DOI: 10.1109/TSE.1987.232894.

[9] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[10] Adrian Frei and Marc Rennhard. "Histogram Matrix: Log File Visualization for Anomaly Detection". In: *2008 Third International Conference on Availability, Reliability and Security*. 2008, pp. 610–617. DOI: 10.1109/ARES.2008.148.

[11] Antti Juvonen, Tuomo Sipola, and Timo Hämäläinen. "Online anomaly detection using dimensionality reduction techniques for HTTP log analysis". In: *Computer Networks* 91 (2015), pp. 46–56. DOI: https://doi.org/10.1016/j.comnet.2015.07.019.

[12] Michael Davis, Weiru Liu, Paul Miller, and George Redpath. "Detecting Anomalies in Graphs with Numeric Labels". In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 1197–1202. DOI: 10.1145/2063576.2063749.

[13] Carmen Cheh et al. "Data-Driven Model-Based Detection of Malicious Insiders via Physical Access Logs". In: *ACM Transactions on Modeling and Computer Simulation* 29.4 (Nov. 2019). DOI: 10.1145/3309540.

[14] Ju Peng Poh, Jun Yu Charles Lee, Kah Xuan Tan, and Eric Tan. "Physical Access Log Analysis: An Unsupervised Clustering Approach for Anomaly Detection". In: *Proceedings of the 3rd International Conference on Data Science and Information Technology*. DSIT 2020. New York, NY, USA: Association for Computing Machinery, 2020, pp. 12–18. DOI: 10.1145/3414274.3414285.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. URL: http://www.deeplearningbook.org.

[16] OWASP Foundation. *Access Control*. 2020. [Online]. URL: https://owasp.org/www-community/Access_Control. Accessed: 2021-03-22.

[17] OWASP CheatSheets Series Team. *Access Control Cheat Sheet*. 2020. [Online]. URL: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html. Accessed: 2021-03-22.

[18] Lauren Collins. "Chapter 11 - Access Controls". In: *Cyber Security and IT Infrastructure Protection*. Ed. by John R. Vacca. Syngress, 2014, pp. 269–280. DOI: 10.1016/B978-0-12-416681-3.00011-2.

[19] William Stallings. "Chapter 4 - Physical Security Essentials". In: *Cyber Security and IT Infrastructure Protection*. Ed. by John R. Vacca. Syngress, 2014, pp. 109–134. DOI: 10.1016/B978-0-12-416681-3.00004-5.

[20] Salvatore Aurigemma and Thomas Mattson. "Privilege or procedure: Evaluating the effect of employee status on intent to comply with socially interactive information security threats and controls". In: *Computers & Security* 66 (2017), pp. 218–234. DOI: 10.1016/j.cose.2017.02.006.

[21] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Computing Surveys* 41 (July 2009). DOI: 10.1145/1541880.1541882.

[22] Thomas M. Mitchell. *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997. ISBN: 9780070428072.

[23] Pádraig Cunningham, Matthieu Cord, and Sarah Delany. "Supervised Learning". In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Ed. by Matthieu Cord and Pádraig Cunningham. Berlin,

Heidelberg, Germany: Springer, Jan. 2008, pp. 21–49. DOI: 10.1007/978-3-540-75171-7_2.

[24] Andrew Y. Ng and Michael I. Jordan. "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". In: *Advances in Neural Information Processing Systems*. Vol. 14. Cambridge, MA, USA: MIT Press, 2001, pp. 841–848. URL: https://proceedings.neurips.cc/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf.

[25] Zoubin Ghahramani. "Unsupervised Learning". In: *Advanced Lectures on Machine Learning*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg, Germany: Springer, 2004, pp. 72–112. DOI: 10.1007/978-3-540-28650-9_5.

[26] "Clustering". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA, USA: Springer, 2017, pp. 226–226. DOI: 10.1007/978-1-4899-7687-1_943.

[27] Vladimir Estivill-Castro. "Why so many clustering algorithms: A Position Paper". In: *SIGKDD Exploration Newsletter* 4.1 (June 2002), pp. 65–75. DOI: 10.1145/568574.568575.

[28] Michail Vlachos. "Dimensionality Reduction". In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA, USA: Springer, 2017, pp. 354–361. DOI: 10.1007/978-1-4899-7687-1_71.

[29] Anke Meyer-Baese and Volker Schmid. "Chapter 2 - Feature Selection and Extraction". In: *Pattern Recognition and Signal Analysis in Medical Imaging (Second Edition)*. Ed. by Anke Meyer-Baese and Volker Schmid. Academic Press, 2014, pp. 21–69. DOI: https://doi.org/10.1016/B978-0-12-409545-8.00002-9.

[30] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. "Dimensionality Reduction: A Comparative Review". In: *Journal of Machine Learning Research* 10 (2009), pp. 66–71. URL: https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf.

[31] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. Sebastopol, CA, USA: O'Reilly Media, Inc., 2019. ISBN: 9781492032649.

[32] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, Dec. 2008. DOI: 10.7551/mitpress/9780262170055.001.0001.

[33] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. "A unifying view on dataset shift in classification". In: *Pattern Recognition* 45.1 (2012), pp. 521–530. DOI: 10.1016/j.patcog.2011.06.019.

[34] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2018. ISBN: 9781491953242.

[35] James Max Kanter and Kalyan Veeramachaneni. “Deep feature synthesis: Towards automating data science endeavors”. In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 2015, pp. 1–10. DOI: 10.1109/DSAA.2015.7344858.

[36] Zahraa S. Abdallah, Lan Du, and Geoffrey I. Webb. “Data Preparation”. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA, USA: Springer, 2017, pp. 318–327. DOI: 10.1007/978-1-4899-7687-1_62.

[37] John T. Hancock and Taghi M. Khoshgoftaar. “Survey on categorical data for neural networks”. In: *Journal of Big Data* 7.1 (Apr. 2020), p. 28. DOI: 10.1186/s40537-020-00305-w.

[38] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. “Explainable Artificial Intelligence: A Survey”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 0210–0215. DOI: 10.23919/MIPRO.2018.8400040.

[39] Ribana Roscher, Bastian Bohn, Marco F. Duarte, and Jochen Garcke. “Explainable Machine Learning for Scientific Insights and Discoveries”. In: *IEEE Access* 8 (2020), pp. 42200–42216. DOI: 10.1109/ACCESS.2020.2976199.

[40] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI”. In: *CoRR* abs/1910.10045 (2019). arXiv: 1910.10045v2. URL: http://arxiv.org/abs/1910.10045v2.

[41] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for Interpreting and Understanding Deep Neural Networks”. In: *Digital Signal Processing* 73 (2018), pp. 1–15. DOI: 10.1016/j.dsp.2017.10.011.

[42] “Loss”. In: *Encyclopedia of Machine Learning and Data Mining*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA, USA: Springer, 2017, pp. 781–781. DOI: 10.1007/978-1-4899-7687-1_499.

[43] PyTorch. *L1Loss*. 2019. [Online]. URL: https://pytorch.org/docs/master/generated/torch.nn.L1Loss.html. Accessed: 2021-05-23.

[44] Ross B. Girshick. “Fast R-CNN”. In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083v2. URL: http://arxiv.org/abs/1504.08083v2.

[45] Lior Rokach and Oded Maimon. “Clustering Methods”. In: *Data Mining and Knowledge Discovery Handbook*. Boston, MA, USA: Springer, 2005, pp. 321–352. DOI: 10.1007/0-387-25465-X_15.

[46] Joe H. Ward Jr. “Hierarchical Grouping to Optimize an Objective Function”. In: *Journal of the American Statistical Association* 58.301 (1963), pp. 236–244. DOI: 10.1080/01621459.1963.10500845.

[47] Ian T. Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202. DOI: 10.1098/rsta.2015.0202.

[48] Ian T. Jolliffe. *Principal Component Analysis*. 2nd ed. New York, NY, USA: Springer, 2002, pp. 1–2. DOI: 10.1007/b98835.

[49] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[50] Geoffrey Hinton and Sam Roweis. "Stochastic Neighbor Embedding". In: *Advances in Neural Information Processing Systems*. Vol. 15. Cambridge, MA, USA: MIT Press, 2002, pp. 857–864. URL: https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[51] Zachary Chase Lipton. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019v4. URL: http://arxiv.org/abs/1506.00019v4.

[52] P. J. Werbos. "Backpropagation Through Time: What It Does and How to Do It". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[53] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[54] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[55] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078v3. URL: http://arxiv.org/abs/1406.1078v3.

[56] Zachary Chase Lipton. "The Mythos of Model Interpretability". In: *CoRR* abs/1606.03490 (2016). arXiv: 1606.03490. URL: http://arxiv.org/abs/1606.03490.

[57] SHAP. *SHAP values*. 2021. [Online]. URL: https://github.com/slundberg/shap. Accessed: 2021-05-24.

A

Appendix 1

A.1 Events

Event name	Frequency
controller.access.card.valid.standard	300,030
controller.door.requestToExit	281,668
controller.door.opened	231,696
controller.door.closed	231,631
controller.reader.validCommonCode	3,473
controller.dac.reader.tamper.restored	2,840
controller.dac.reader.tamper.active	2,818
controller.dac.communicationFailure.restored	2,745
controller.door.forcedOpen	2,304
controller.dac.communicationFailure.active	2,229
controller.door.mode.locked	2,177
controller.door.mode.unlocked	2,173
controller.door.notClosed	2,016
controller.access.card.invalid.door	1,357
controller.access.card.invalid.standard	768
system.invalidCardInLcuThatShouldBeValid	680
controller.dac.tamper.active	422
controller.hio_node.communicationRestored	402
controller.door.mode.buzzerEnabled	327
controller.door.mode.exit.pinCardNumber	238
controller.door.mode.exit.modePinRequired	159
controller.door.mode.access.pinOnlyAllowed	134
controller.access.card.invalid.inhibited	117
acs.door.update	115
acs.doormodetemplate.update	113
controller.dac.tamper.restored	109
controller.door.mode.access.modePinRequired	79
controller.door.unlock	60
controller.door.mode.maintainedUnlock	54
controller.door.forcedUnlock	52
controller.dac.powerOn	50

controller.door.mode.exit.buyAlarmtime	49
controller.dac.digitalOutputChanged	42
controller.hio_node.communicationFailure	41
controller.door.pulseOpen	35
acs.authorization.create	14
controller.access.card.invalid.format	13
acs.doormodetemplate.create	12
controller.dac.inputChanged	7
controller.door.mode.exit.pinOnlyAllowed	6
controller.door.mode.maintainedLock	6
acs.door.delete	6
acs.authorization.delete	3
controller.HubRestartedAfterReset	3
controller.door.mode.motorlockLowSecurity	3
controller.door.mode.rteMaintainedUnlock	2
controller.door.mode.rteEnabled	2
acs.doormodetemplate.delete	2
acs.accessarea.update	1
acs.accessarea.delete	1

Table A.1: Event distribution. The total number of events is 1,073,284. There are 50 types of events in total.