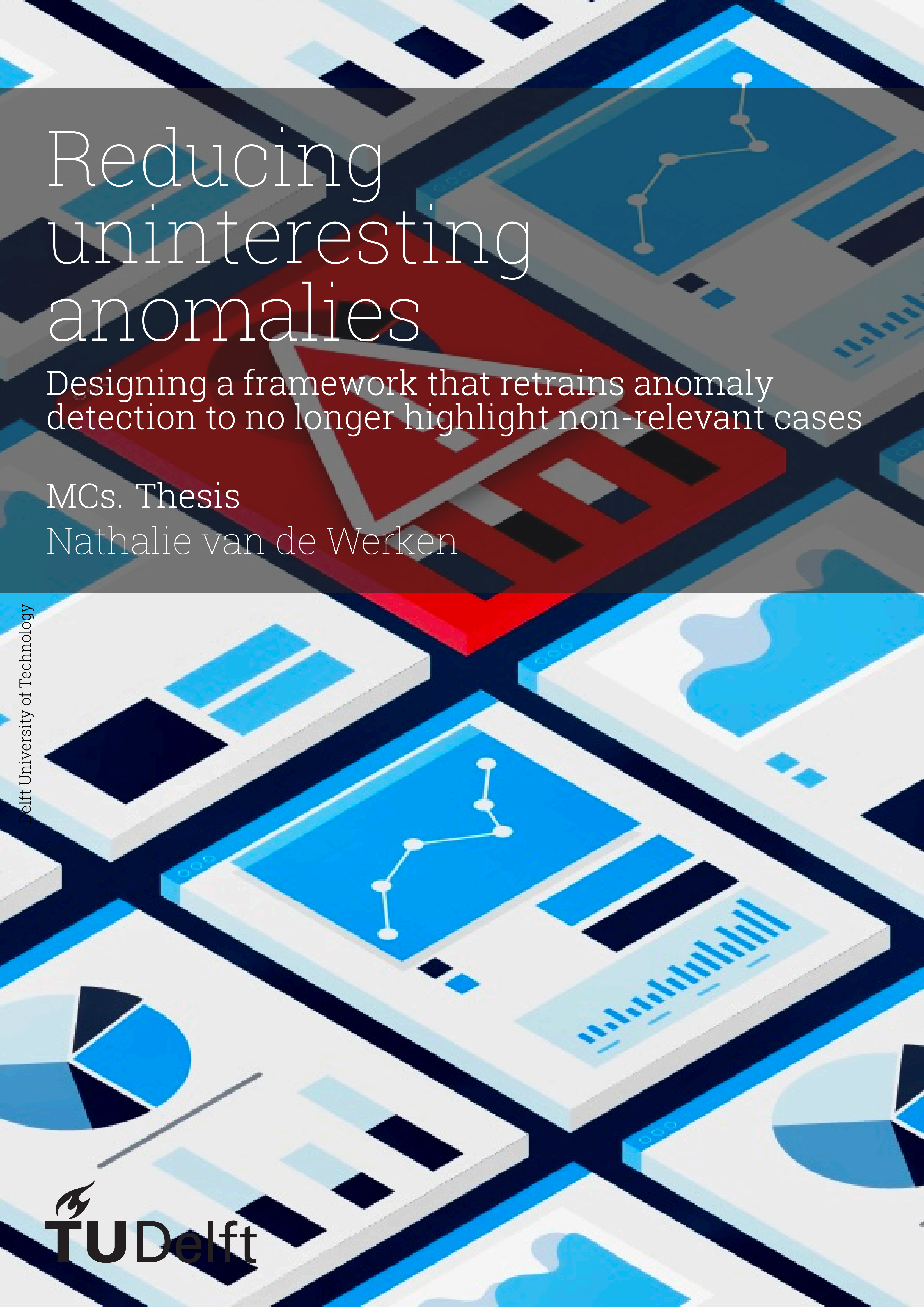


Reducing uninteresting anomalies

Designing a framework that retrains anomaly detection to no longer highlight non-relevant cases

MCs. Thesis

Nathalie van de Werken

Delft University of Technology

Reducing uninteresting anomalies

Designing a framework that retrains anomaly detection to no longer highlight non-relevant cases

by

Nathalie van de Werken

Student number:	5064759
Project Duration:	March, 2024 - May, 2025
Thesis Committee:	Anna Lukina
	Emir Demirović
	Luciano Cavalcante Siebert
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover:	StrongDM (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

The field of anomaly detection has long fascinated me due to its broad applications in fields ranging from cybersecurity to healthcare. Here, I always found it fascinating how despite it being an unsupervised problem, it still performs so well. However, I also saw that quite often there are many different types of anomalies, and I wondered how interesting all of those were for the end user of the system. I especially considered the situation where the user keeps on getting alerted for anomalies that are not interesting. This realization sparked the core motivation behind this thesis—how can we refine anomaly detection systems to better align with what the end user wants and only highlight the anomalies they deem interesting?

This thesis is the culmination of months of research, experimentation, and problem-solving. It would not have been possible without the guidance and support of many people. First and foremost, I would like to express my deepest gratitude to my supervisor, Anna Lukina, for their invaluable insights, constructive feedback, and unwavering encouragement throughout this project. I would also like to thank my second supervisor, Emir Demirović, for their fresh look on this work every time. Additionally, I would like to say a special thank you to my family and friends for their patience and support during the long hours spent developing and refining this work.

Finally, I hope this research contributes to the growing field of anomaly detection by providing a practical approach to reducing irrelevant anomalies. I am excited to see how future developments build upon these findings to further enhance intelligent detection systems.

Nathalie van de Werken
Delft, April 2025

Abstract

Anomaly detection is a cornerstone of data analysis, aimed at identifying patterns that deviate from expected behaviour. However, conventional anomaly detection methods often fail to differentiate between actionable anomalies and those that, while statistically anomalous, are irrelevant to the user's goals. Such uninteresting anomalies, originating from distinct, unrelated distributions, contribute to false alarms and resource inefficiencies, particularly in critical domains like cybersecurity and healthcare. This thesis proposes a novel, adaptive framework that retrains anomaly detection models to exclude uninteresting anomalies from being flagged, thereby improving the relevance of detected anomalies.

Central to this framework is the use of the Synthetic Minority Over-sampling Technique for Nominal and Continuous (SMOTE-NC) to artificially augment datasets with labelled uninteresting anomalies, transforming them into regular data. The framework also incorporates user feedback to iteratively refine model performance during deployment. A key finding of this research is that the effectiveness of the framework is highly dependent on the degree of distinguishability between interesting and uninteresting anomalies. Specifically, when the two types of anomalies are clearly distinct in terms of their statistical and categorical features, the framework achieves a significant reduction in false positives without adversely affecting the detection rate of actionable anomalies or the accuracy of regular data.

The framework was evaluated using four state-of-the-art anomaly detection models: Isolation Forest, One-Class Support Vector Machines, Autoencoders, and Variational Autoencoders. It was then tested using two datasets: one in cybersecurity, involving various attack types, and another in healthcare, where anomalies represent different diagnostic categories. The results demonstrate that the framework can effectively identify and suppress uninteresting anomalies, achieving over 90% accuracy in classifying these cases as regular data under favourable conditions. Notably, when the distinction between interesting and uninteresting anomalies was substantial, the models retained their ability to detect actionable anomalies, with minimal degradation in overall accuracy. Furthermore, it was seen that a significant number of samples are needed to be able to successfully represent the uninteresting anomalies class, a minimum of around 50. When not using the framework, it takes many more samples for the algorithm to successfully no longer detect these uninteresting anomalies as anomalies. This class then needs to make up a significant amount of the training data, so depending on the size of the training data set, this can mean thousands of samples. Gathering around 50 samples poses no problem for the framework, as it is meant to be especially relevant when we have too many uninteresting anomalies, where it is constantly giving false alerts, so there is usually a sufficient number of samples available.

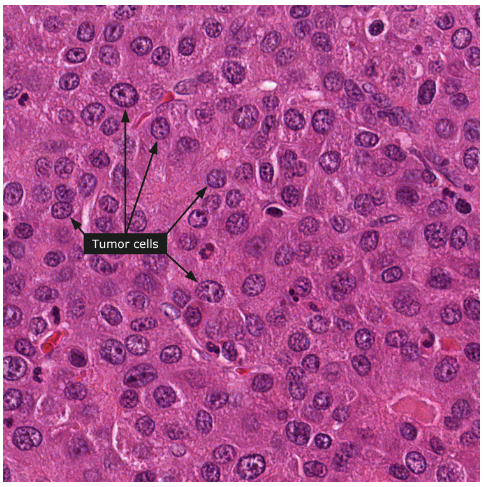
Contents

1 Introduction

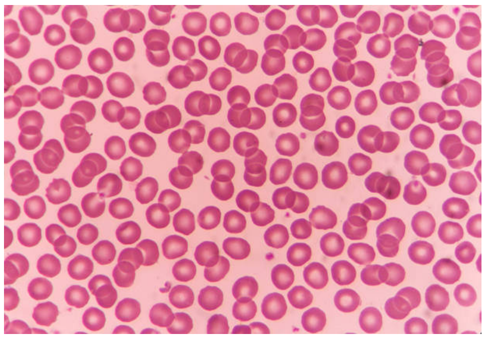
Anomaly detection is a critical process in data analysis that involves identifying patterns in the data that do not conform to the expected behaviour [8]. These atypical patterns, which are known as anomalies, can indicate significant and often actionable information. Detecting these is often a challenging task due to the diverse nature of anomalies and their tendency to be context-specific. Anomalies can manifest as sudden spikes or drops in values, unexpected sequences, or deviations from statistical properties. Their detection requires sophisticated algorithms and techniques capable of handling large volumes of data with high dimensionality [2]. The complexity is further compounded by the need to minimise false positives, where normal data is mistakenly identified as anomalous, and false negatives, where actual anomalies go undetected. Addressing this problem involves developing robust models that can learn the intricate patterns of normal behaviour and accurately identify deviations, even in the presence of noise and variability in the data. This has many real-world applications, e.g. in cybersecurity [25], to monitor if there are irregularities in certain metrics which might indicate that the system is under attack. Another use case is in the medical world, where it can be used to detect if certain cells are anomalous and further testing is required [20].

An ongoing challenge in anomaly detection is the inability to distinguish between different types of anomalies. When an anomaly is defined as a data sample that does not fit in, encountering one does not always mean that it is an interesting case that needs to be looked at. For instance, when it comes to network monitoring, the servers being down for maintenance is an anomaly, as it does not happen very often, but it does not require an entire procedure to protect the servers against an attack. In the medical domain, an algorithm has been trained to distinguish cancerous liver cells from healthy liver cells, but during deployment of the algorithm, it turns out that the samples might also contain blood cells. The user of the model does not care about the anomaly if it is caused by an incidental blood cell. These are all novel classes; classes that were not present in the training data, but are not of interest to the user.

Novel classes usually encompass uninteresting anomalies, and thus, we want to ensure that an alarm is only raised in the case of a real anomaly. A big challenge here is that anomaly detection is an unsupervised problem, so the user cannot simply tell the algorithm that it is wrong and retrain it based on this new sample. Turning it into a supervised problem is possible, however, this means that all the training data needs to be labelled, which is very time-intensive and not always feasible. As of now, the field of detecting true anomalies and false positives has seen a severely limited amount of research. Some approaches have been found that rely on neural networks [54]. They ask a user to label how anomalous they think the sample is, and combine this with how certain the algorithm is that it is an anomaly and send this back to retrain. This is an interesting approach, but it relies on neural networks, which are big black boxes when it comes to how the model has been formed. Therefore, they are lacking in terms of explainability. Explainability is especially important in anomaly detection, as this allows the user to trust the system and feel comfortable using it [34]. However, it is of even more importance in this use case, as the user is guiding the model, so we need to ensure that the user does not make a mistake and label an interesting anomaly as uninteresting. In addition to the lack of



(a) Liver cells and cancerous liver cells [14]



(b) Red blood cells [31]

Figure 1.1: There is only a small difference between the regular data (healthy liver cells) and the interesting anomalies (cancerous liver cells), as can be seen in **(a)**, but the uninteresting anomalies (blood cells) look completely different in **(b)**

explainability, it requires feedback on every sample by the user, which can become a bottleneck.

For simplicity, we will introduce a real-world example to make the problem more concrete, which will be referred back to throughout the paper. In this example, there is an anomaly detection model that analyses liver cells and tries to detect cancerous cells, the anomalies in this case. After this model has been trained, however, it is discovered that the samples that are submitted to the system from the tests can contain blood cells. Blood cells are indeed anomalous and look very different from both the normal and the cancerous cells, and therefore are detected as anomalous. In Figure 1.1, you can see that the cancerous liver cell mostly looks significantly different in size, but the blood cell looks completely different in all of these attributes. This will raise many false alerts, and as a result, the domain expert, in this case, the physician, will have to look at many more cases. They might even disregard the entire model as they deem it inaccurate and ineffective. The cells have several features, such as the size of the cell, colour, number of spikes, and number of entrances to the cell. Due to this, it is easily distinguishable from both the interesting anomaly and the regular data class.

The objective of this thesis is to investigate how to continually adapt an anomaly detection algorithm while it is in production to ensure it does not raise an alarm anymore for anomalies which are of no interest to the user. This thesis will contribute to the scientific community in three significant ways. First of all, we will define the problem of no longer detecting uninteresting anomalies as a new challenge in anomaly detection. Secondly, we will show a framework that will be able to solve this challenge while the model is in production, with the help of a user, and retrain the model such that these uninteresting anomalies will no longer be flagged. Finally, we have implemented this framework, which uses SMOTE, a resampling algorithm, to create more samples of this uninteresting class. Then the model will learn to see these as part of the regular data and no longer raise these alarms while still being effective in detecting actually interesting anomalies. We have tested this framework using 4 different models, which have all been used in anomaly detection before. These are isolation forests (IForest), one-class support vector machines (OCSVM), autoencoders (AE) and variational autoencoders (VAE). These models were then tested on two datasets, one in the domain of cybersecurity and the other in the medical domain. Here, we found that we can greatly increase the number of uninteresting anomalies that will no longer be labelled as anomalies as long as these classes are very distinct. This does not mean that none of them will slip through anymore, as it was found that it is difficult to get above a 90 % accuracy in the uninteresting class. However, the rate of regular data being classified correctly can decrease by a relatively small amount, as well as anomalies being classified correctly. Furthermore, it is seen that only around 50 samples are needed to get good results, but gathering more samples of the uninteresting class will lead to its accuracy increasing.

1.1. Research questions

This all comes together to form the main research question:

> How can we design a framework such that we can mitigate the effect of novel classes during anomaly detection?

To answer this research question, we will answer the following subquestions:

1. How do we detect a novel class compared to an anomaly, and help the user see that?
2. How can we retrain the model on the fly to ensure that it no longer detects said novel classes as anomalies?
3. How can we test the effectiveness of this framework?

1.2. Thesis structure

This thesis is structured as follows. First, we will go through all the relevant related work to this problem in chapter 2, and the background of work which has been directly built on top of for this thesis in chapter 3. After this, we will formally define this new problem in chapter 4. Following this, in chapter 5, we will talk about the technical steps we have taken in this thesis. In the first section, we will discuss the overall framework, and how SMOTE has been used to implement this framework is discussed in the second section. In chapter 6, we will go through the setup that has been used for the experiments, as well as for each experiment, describe how we conducted them. After this, in chapter 7, we will display the results that have been acquired and analyse these. Finally, in chapter 8, we will discuss recommendations for future work and end with the conclusion in chapter 9.

2 Related work

Before we can start answering the research question, we have to do research into the current literature on anomaly detection. Therefore, we will go through all the relevant background information in order to understand what the state of the art is in anomaly detection in section 2.1, explainable anomaly detection in section 2.2, and active anomaly detection in section 2.3. Finally, we will also discuss the similarities to zero-shot learning and active learning in section 2.4.

2.1. Anomaly detection

There are many different approaches to anomaly detection [8]. These can range from approaches that do not require any machine learning and instead work algorithmically, to deep learning. The first approach that was considered is the isolation forest [35]. In this technique, we build several trees that cluster a group of data together by taking subsets of the samples by splitting on random features. This entire process does not require much memory. After that, the anomaly detection works as follows: we see for every tree how many splits it takes for it to no longer be part of the cluster. The anomalies will have a relatively low score and be labelled as an anomaly. An advantage of this approach is that you do not need to label the input data. This is because it looks at anomalies based on how isolated they are from the rest of the data. Many extensions have been proposed, such as deep isolation forests [51], which use deep learning to no longer use random splits and therefore can capture the more complex underlying distribution. Another approach is to use sliding windows to account for data changing over time [16]. This approach also conveys an intuitive way of looking at anomaly detection, and therefore, it leads to an understandable result while still being complex enough to deal with the more subtle aspects of the data.

Some statistical models try to detect an anomaly based on the z-score. The z-score represents how many standard deviations a data point is from the mean of a dataset, and samples that lie furthest from the mean are considered anomalies. This method is very straightforward, therefore allowing for easily interpretable and robust results. As such, it has been used for a long time. It was first introduced in 2012 by Ferragut et al. [21], and it was later applied to time series as well [55]. This approach does have its shortcomings, as it is a rather minimalistic approach. It is often combined with other, more complex techniques to achieve better performance [11]. There are also approaches that use more complicated statistical analysis, where they use hybrid weights to make the model more robust against outliers [53].

Another branch of anomaly detection algorithms uses neural networks to detect anomalies. This is what most new research focuses on, with lots of different extensions and different neural networks being proposed. However, a big downside of this technique is that the inner workings of neural networks remain difficult to explain. Here, again, there are many possibilities, from using one-class neural networks [7] to using convolutional neural networks [30]. Under this branch also fall autoencoders and variational autoencoders; for further explanation on how these work, see sections 3.4.1 and 3.4.2.

2.2. Anomaly detection with Explainable AI

A large area of research within the field of anomaly detection is extensions of anomaly detection that make the models more understandable to the user. This is relevant to this research, as the proposed framework will heavily rely on user feedback, so we must minimise the probability of user error. A way to do that is to make the user understand the reasons behind a decision, so they can judge if it was reasonable.

There are two different options when it comes to making anomaly detection more understandable. We can work on making the overall model more explainable, which is known as transparent design. Alternatively, we can focus on the individual samples and explain why they were marked as anomalies or not. These are known as post-hoc explanations [50]. All of these approaches are very specific, as they heavily depend on the technique that is being used in anomaly detection. In this research, we will focus on explaining the algorithm per sample, as we want to explain to the user why the specific sample has been marked anomalous.

The simplest approach to explain anomaly detection is the heuristic approach. Here, we do not use machine learning for detecting anomalies. Instead, it requires a simple algorithm that emulates the black-box setting, such as a decision tree or one-layer neural network [22]. This allows for simplicity in the model, but it has many drawbacks. First of all, the model cannot become better than a human agent, because a user picks the values for these models, and therefore it will not be able to pick up on patterns the human does not realise exist. It also requires some very specific domain-level knowledge to be put into the model, specifically, as can be seen in research by Brociek et al. [4]. Nevertheless, this is still a simple method that gives clear explainability, as you can handpick which part of the algorithm works for which feature. A notable benefit is that this needs significantly less data to test compared to big models. For the purposes of this research, however, this approach is not preferable, because we want to work with more complex models, as these tend to lead to higher accuracy.

As mentioned in section 2.1, autoencoders are state-of-the-art when it comes to deep learning anomaly detection, and there is also an extension for it that makes it more understandable. In this approach, we use an extra formula that specifies for each feature how important it was in detecting the anomaly. It calculates how much the feature deviates from the mean given its standard deviation, and also normalises it to ensure that binary features do not skew the results [38]. Another paper talks more about giving explainable results when we are using neural networks for explainable AI, by using an attention mechanism which signifies which part of the input data has been used to give the result [5].

2.3. Monitoring anomaly detection

The state of the art still considers anomaly detection a mostly static topic, where not a lot of research has been done on retraining the model over time. Some research has been done about retraining the network based on normal outliers, however, it works with a completely different technique of detecting anomalies, namely using elastic weight retraining to change where the boundary is, which is a much less advanced algorithm [26] and a lot less versatile than our technique will be. Another paper focuses on unlearning, a process in which the objective function is changed in such a way that it allows for a flexible model but also prevents forgetting of the data that has been learned [18]. This is different from our approach, as we do not focus on how it shifts over time. In a third paper, an approach using neural networks was used, and they retrained the model based on the new samples gathered in real life [49]. However, no user input is used for that, so it cannot gain insights from a domain expert as our approach can. There is also a fourth paper that looks at uninteresting anomaly detection, but they turn it into a completely supervised problem as they do post-processing to change the order in which the user gets to see the anomalous samples [41], not changing or retraining the model.

2.4. Similarities to zero-shot learning and active learning

In zero-shot learning, we focus on a learner who has to observe samples from classes which were not observed during training and needs to predict the class that they belong to [9]. Even though the concept is fairly similar to what we are investigating, the approach is very different, as it usually needs a description of the other class, something our framework will not need [45]. In addition, current approaches have mostly been tested on computer vision models, and not for anomaly detection, but some extensions to natural language processing have been researched [6]. Because of this and the need for descriptions, the approach is significantly different.

In active learning, the user is queried while the system is in use, such that it can be retrained to ensure that it is still accurate [47]. This is exactly what we will be using in this paper. However, the exact approach we are taking has not been studied before. This is because it always requires the user to label the data and then retrain based on that, but since anomaly detection is an unsupervised problem at its core, this is not possible. It has been applied before in anomaly detection, but all of these still rely on labelling a subset of the data [46], not just the relevant cases, and work with deep neural networks, which are lacking in terms of explainability [54] [43].

3 Background

We will design a framework to solve the problem of no longer detecting the uninteresting anomalies as anomalous. However, this has not been done from scratch and is instead based on a combination of previous approaches. In this chapter, we will go through all these works. In section 3.1, we will explain the basics of the resampling method we use in this paper. Then we will talk about the paper whose framework we are using for the monitoring in section 3.2, and discuss the method that we are using for anomaly detection in detail in section 3.3. Finally, in section 3.4, the theory behind all the models will be explained.

3.1. Resampling of data

Our work will heavily rely on resampling data of the new anomalous class that does not have nearly as many samples. We have the choice between undersampling the majority class and oversampling the minority class. In undersampling, you take the majority class and drop a part of the samples to make it a similar size to the minority class. However, this tends to lose valuable information, and if there are very few cases of the other class, this is still impossible. In addition, many modern models used in anomaly detection are deep models, which benefit from having more training data. Therefore, we will be using one of the most commonly used oversampling techniques, SMOTE [10]. This technique has shown promising results in biomedical applications [27] and in cybersecurity [1] [13].

3.1.1. SMOTE

Synthetic Minority Over-sampling Technique, known as SMOTE, is a widely used method for addressing class imbalance in classification problems. Instead of simply duplicating existing instances from the minority class, SMOTE generates new synthetic examples by interpolating between existing ones. For a visual overview of the framework, see figure 3.1.

More specifically, for each sample in the minority class, the algorithm identifies its k nearest neighbours, typically based on Euclidean distance. Then, it randomly selects one of these neighbours and creates a synthetic data point by taking a convex combination of the two. This new point lies somewhere along the line connecting the original instance and its selected neighbour. As a result, SMOTE generates samples similar to existing ones, but not identical, helping to reduce overfitting and improve generalisation.

This approach has the advantage of respecting the local structure of the data and preserving potential correlations between features. However, it can also introduce synthetic samples in regions where the classes overlap, which may lead to less clear decision boundaries, especially if there is no actual correlation between features.

There are also some extensions to the SMOTE procedure. These focus on which samples are used to regenerate the new samples [19], where most choose to take samples where data is dense [42], or data that is close to the decision boundary [24].

We are choosing to focus on the most basic implementation. Since the number of available samples is limited for the minority class, choosing which points to focus on is not as relevant, and it cannot benefit the framework a lot. Due to the specific implementation of SMOTE, by default, it needs the data to be labelled, as we need to supply it with at least two classes. This means that for our implementation, the two classes are the regular data and the uninteresting anomalies, to ensure that these will be resampled.

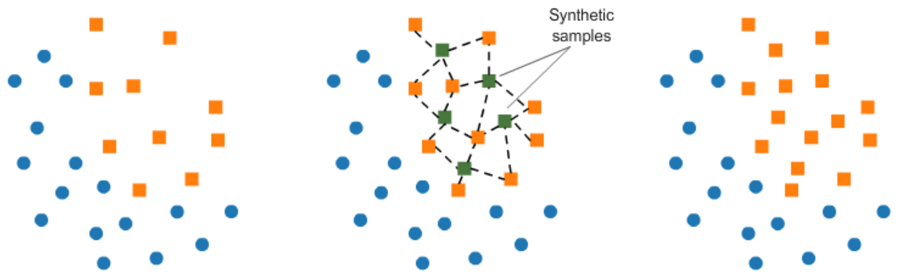


Figure 3.1: Illustration of the SMOTE process [12]. The left panel shows an imbalanced dataset with a minority class (orange squares) and a majority class (blue circles). The middle panel demonstrates the creation of synthetic samples (green stars) between existing minority instances. The right panel displays the resulting balanced dataset after oversampling.

3.2. Framework feedback

We have based our way of looking into the continuous monitoring of an anomaly detection algorithm on that of research on active monitoring of neural networks by Kueffner et al. [29]. The paper introduces an active monitoring framework for neural networks to address challenges in maintaining accuracy in dynamic environments. The framework operates in parallel with the neural network, interacting with a human user for incremental adaptation. It includes a quantitative monitor to improve precision and detect inputs from novel classes. By distinguishing between known and unknown novelties, the framework adapts the neural network and monitors at prediction time, either by learning new classes or retraining with updated information. Here, the model detects novelties by looking at the output of the penultimate layer and seeing if the distance between this and the average of its class is significant enough. If this value is indeed anomalous, it is sent to a domain expert such that they can see if it is indeed an anomaly, and if the system made a mistake by reporting a false positive, it can be retrained. Also, if it is a new class, they can label it as such and retrain the system so it will be more accurate when detecting anomalies. This adaptive approach aims to prevent catastrophic forgetting and system failures caused by novel input classes. Figure 3.2 shows an overview of this framework. Experimental evaluation on diverse benchmarks confirms the framework's benefits in dynamic scenarios. The framework enhances the transparency and interpretability of neural network operations, facilitating informed decision-making in real-world applications.

This is an interesting approach, but we modified it such that it can work for an anomaly detection algorithm. This means it cannot use an anomaly detection algorithm to detect the difference between a novel class and the regular data, as it is already running an anomaly detection algorithm tasked to continue labelling interesting anomalies as anomalous. In this approach, all anomalies are given to the user to say which novel class they belong to. This is different from our approach, as even though we do ask the user as well, our user only needs to respond when they deem an anomaly as uninteresting. Here, they do not need to give a label to what kind of uninteresting it is. This is to further reduce the human bottleneck. Also, there is a lot less freedom in this approach as it only works for neural networks, so we will be extending it and creating a generic framework.

3.3. Anomaly detection

The anomaly detection algorithm that we opted to use is based on a paper focusing on interpreting unsupervised anomaly detection by Li et al. [33]. They describe an algorithm that explains how it decides by making hypercubes of the data points and showing the decision boundaries. The technical steps involve distribution decomposition rules to simplify the complex distribution of normal data into multiple compositional distributions. An unsupervised Interior Clustering Tree (IC-Tree) is designed to incorporate model predictions into the splitting criteria, aiding in identifying rules that define normal data and

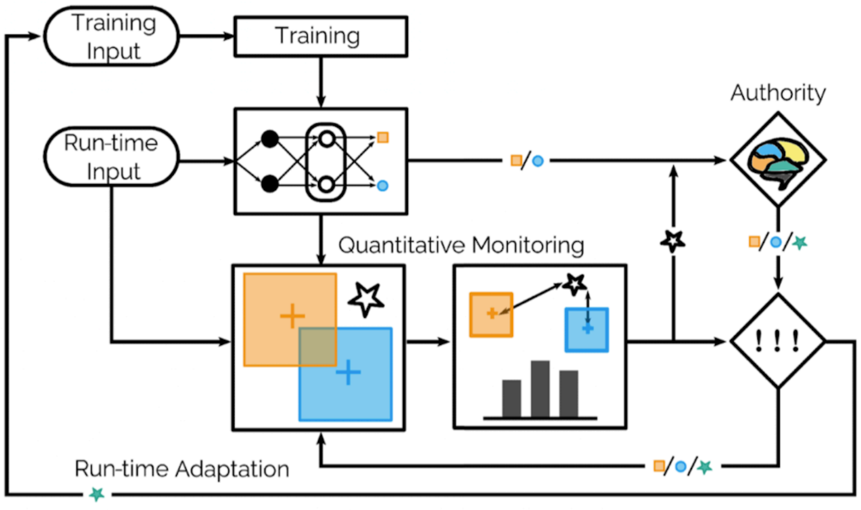


Figure 3.2: The architecture of an active monitoring system for neural networks [29], showing its main components and data flow. At the top left, incoming data samples are processed by the neural network, with predictions produced alongside feature representations from the penultimate layer. These representations are passed to a quantitative monitor (middle), which compares them against class-specific reference clusters using a distance metric. If the distance exceeds a class-dependent threshold, the prediction is flagged as potentially novel. Flagged samples (right) are routed to a human authority for inspection, either confirming a false positive or identifying a new class. The result informs downstream actions: the monitor may be refined, or the neural network retrained to incorporate new knowledge.

anomalies. The Compositional Boundary Exploration (CBE) algorithm is then used to obtain boundary inference rules that estimate the decision boundary of the original model on each compositional distribution, enhancing understanding of the decision boundaries of the black-box model. By merging the distribution decomposition rules and boundary inference rules into a rule set, the method presents the inferential process of the unsupervised black-box model in a human-understandable way. For an overview of this process, see figure 3.3. Comprehensive experiments on four distinct unsupervised anomaly detection models using real-world datasets show superior performance in terms of fidelity, correctness, and robustness compared to existing methods.

This model will be used and further extended such that it will be compatible with the framework of section 3.2.

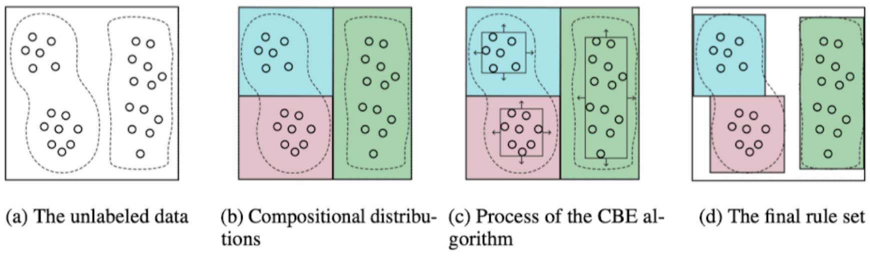

(a) The unlabeled data
(b) Compositional distributions
(c) Process of the CBE algorithm
(d) The final rule set

Figure 3.3: Illustration of the interpretability framework for unsupervised anomaly detection introduced by Li et al. [33] Here, you can see in (a) the original unlabeled dataset with clusters of normal data. In (b), the complex data distribution is decomposed into multiple compositional distributions. Then, in (c), the Compositional Boundary Exploration (CBE) algorithm identifies decision boundaries for each compositional region using hypercubes. Lastly, in (d), you can see the final rule set combines distribution decomposition and boundary inference rules to approximate the decision boundaries of the original black-box model in an interpretable form.

3.4. Models used

To test the effectiveness of the designed framework, it will be tested on four different models. These models have been chosen as they are all state-of-the-art in anomaly detection. They all rely on different principles, and a brief overview of how they work will be given in this section. Extensions of these models are possible, which can achieve even better results, but these are also more specific per use case. As we first of all want to have a good understanding of whether this framework can work, we have opted for the basic versions of the models.

3.4.1. Autoencoders

Autoencoders are neural network-based models used for unsupervised anomaly detection by learning a compressed representation of normal data. They consist of two main components: an encoder f_θ that maps an input $\mathbf{x} \in \mathbb{R}^n$ to a lower-dimensional latent representation $\mathbf{z} \in \mathbb{R}^m$ (where $m < n$), and a decoder g_ϕ that attempts to reconstruct the original input from this latent code, i.e., $\hat{\mathbf{x}} = g_\phi(f_\theta(\mathbf{x}))$.

The model is trained by minimising a reconstruction loss, typically the Mean Squared Error (MSE), given by:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

During training, the autoencoder learns to reconstruct regular data accurately. At inference time, anomalies, which differ significantly from the training distribution, tend to produce larger reconstruction errors, since the model fails to capture their structure in the latent space. By thresholding this reconstruction error, we can flag inputs as anomalous.

Autoencoders are highly flexible and capable of modelling complex, nonlinear relationships in high-dimensional data, making them useful in applications like fraud detection, manufacturing fault diagnosis, and medical imaging. However, their performance depends heavily on factors such as the architecture (e.g. number of layers, size of the bottleneck), choice of loss function, and use of regularisation techniques like dropout or weight decay. They require a representative sample of normal data for training, and their sensitivity to anomalies in the training set or overfitting can limit robustness in practice. See Figure 3.4 for a visual overview of the autoencoder architecture.

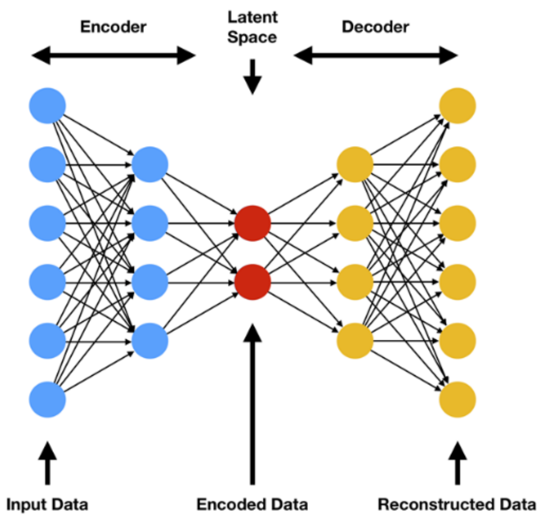


Figure 3.4: The architecture of an Autoencoder [15]. Here we see the two stages of the Autoencoder, where we first encode the data through multiple layers, after which we try to recreate it using the decoder.

3.4.2. Variational Autoencoders

Variational Autoencoders (VAEs) are a probabilistic extension of traditional autoencoders used for anomaly detection, with a key difference in how they model and generate data. Instead of learning a deterministic mapping, VAEs learn a latent distribution over the input space by approximating the true data-generating probability density.

The encoder maps each input $\mathbf{x} \in \mathbb{R}^n$ to the parameters of a probability distribution in latent space, typically a multivariate Gaussian. That is, it outputs a mean $\boldsymbol{\mu}(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma}(\mathbf{x})$, which define a latent distribution $q_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{x})))$. The decoder samples a latent variable $\mathbf{z}$ from this distribution and reconstructs the input: $\hat{\mathbf{x}} = g_\phi(\mathbf{z})$.

Training VAEs involves minimising a composite loss function that balances two terms:

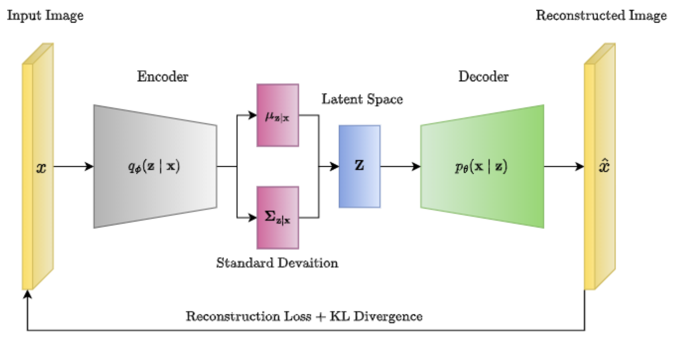
- A reconstruction loss, the expected negative log-likelihood $\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[-\log p_\phi(\mathbf{x}|\mathbf{z})]$, which ensures the output resembles the input.
- A regularisation term, the Kullback-Leibler (KL) divergence, which encourages the approximate posterior to stay close to a prior distribution $p(\mathbf{z})$, usually $\mathcal{N}(0, \mathbf{I})$: $D_{\text{KL}}\left(q_\theta(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right)$

The total loss is thus:

$$\mathcal{L}_{\text{VAE}}(\theta, \phi; \mathbf{x}) = \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[-\log p_\phi(\mathbf{x}|\mathbf{z})] + D_{\text{KL}}\left(q_\theta(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right)$$

This probabilistic framework allows VAEs to model uncertainty and detect anomalies either through low likelihood under the learned latent distribution or via high reconstruction error. For an overview of the architecture of a VAE, see figure 3.5. VAEs are well-suited to capturing complex data distributions and offer a principled method for generating realistic data samples, making them applicable in domains such as fraud detection, image anomaly detection, and scientific data modelling.

However, VAEs require careful tuning of latent dimensionality, loss weighting, and architecture to balance reconstruction fidelity and latent space regularity. If the KL divergence term dominates during training, reconstructions can degrade, especially for fine-grained or high-resolution data. Additionally, the stochastic sampling and variational inference steps introduce added computational complexity compared to standard autoencoders.



Input Image
Reconstructed Image
Encoder
Latent Space
Decoder
x
$q_\phi(\mathbf{z} \mid \mathbf{x})$
$\mu_{\mathbf{z}|\mathbf{x}}$
$\Sigma_{\mathbf{z}|\mathbf{x}}$
$\mathbf{z}$
$p_\theta(\mathbf{x} \mid \mathbf{z})$
$\hat{x}$
Standard Devaition
Reconstruction Loss + KL Divergence

Figure 3.5: The architecture of a Variational Autoencoder [48]. The encoder maps input data to the mean and standard deviation of a latent Gaussian distribution. A latent variable is sampled and passed through the decoder to reconstruct the input. Training minimises both reconstruction loss and KL divergence, enabling the VAE to model uncertainty and detect anomalies.

3.4.3. Isolation Forest

Isolation Trees are a tree-based method for anomaly detection that works by recursively partitioning the feature space using randomly selected features and split values. At each internal node, a feature is chosen uniformly at random, and a split point is selected uniformly within the range of that feature. This process continues until the data point is isolated in a leaf node. Since anomalies are typically rare and differ significantly from normal points, they are more likely to be isolated earlier in the process, requiring fewer splits and resulting in shorter path lengths.

An Isolation Forest is an ensemble of such isolation trees. The anomaly score for a data point $\mathbf{x}$ is computed based on the average path length $h(\mathbf{x})$ across all trees in the forest. Formally, the anomaly score is defined as:

$$s(\mathbf{x}, n) = 2^{-\frac{h(\mathbf{x})}{c(n)}}$$

Where $h(\mathbf{x})$ is the average path length of $\mathbf{x}$, n is the number of samples used to build each tree, and $c(n)$ is the average path length of unsuccessful searches in a Binary Search Tree, given by:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

with $H(i)$ being the i-th harmonic number $H(i) \approx \ln(i) + \gamma$, and γ is the Euler-Mascheroni constant ($\gamma \approx 0.5772$).

A higher score (close to 1) indicates that a point is likely to be anomalous, while a lower score (close to 0) suggests normality. Isolation Forests are computationally efficient, with a time complexity of $O(n \log n)$, and do not rely on distance metrics or density estimates, making them well-suited for high-dimensional and large-scale datasets. In figure 3.6, you can see a visual representation of the architecture of an Isolation Forest.

However, the performance of Isolation Forests can be sensitive to hyperparameters such as the number of trees and their maximum depth. Moreover, the method assumes that anomalies are more susceptible to isolation, which may not hold in cases with complex feature interactions or clustered anomalies. In such cases, additional preprocessing or hybrid models may be necessary to improve detection performance.

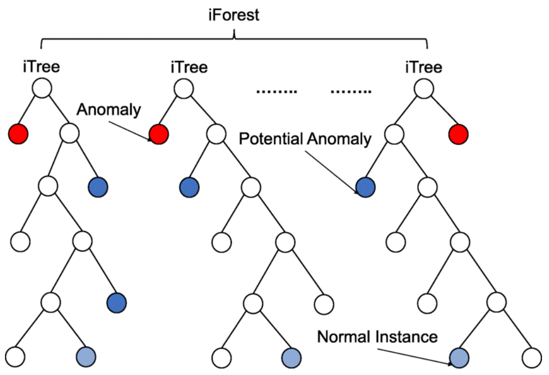


Figure 3.6: The architecture of the Isolation Forest [44]. You can see that it is made out of multiple Isolation Trees. To test a sample, the average height is computed, and if this is under a threshold, it will return anomalous.

3.4.4. OCSVM

The One-Class Support Vector Machine (OCSVM) is a kernel-based anomaly detection technique that identifies anomalies by learning a decision boundary that encompasses the majority of normal data in a high-dimensional feature space. It is based on the idea of separating data from the origin with maximum margin in a transformed feature space, using the kernel trick to handle non-linear patterns.

Given a set of training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^d$, OCSVM solves the following optimization problem:

$$\min_{\mathbf{w},\rho,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n} \xi_i - \rho$$

subject to

$$(\mathbf{w} \cdot \phi(\mathbf{x}_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \dots, n$$

Where $\phi(\cdot)$ is the mapping function to a high-dimensional feature space, $\nu \in (0, 1]$ controls the trade-off between the fraction of outliers and the margin, and ξ_i are slack variables allowing some data points to lie outside the decision boundary.

The decision function is then given by:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho\right)$$

Where $K(\mathbf{x}_i, \mathbf{x}) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle$ is a kernel function, typically the Radial Basis Function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

Points for which $f(\mathbf{x}) < 0$ are considered anomalies, as they lie outside the learned boundary. For an overview of this method, see figure 3.7. OCSVM is particularly effective for capturing complex, non-linear structures in data and is widely used in applications such as fraud detection, intrusion detection, and medical data analysis.

However, OCSVM's performance is sensitive to the choice of kernel parameters (e.g. γ), the regularisation parameter ν, and the proportion of anomalies assumed in the data. Moreover, the algorithm involves solving a quadratic programming problem, making it computationally expensive for large-scale datasets. It is also prone to reduced robustness if the training data contains outliers or is not well-representative of the normal class.

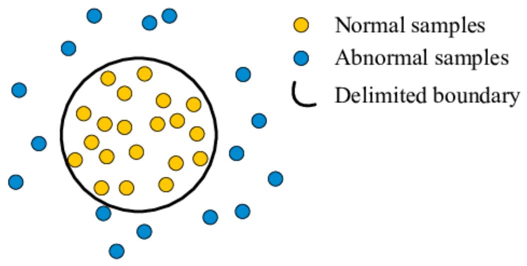


Figure 3.7: The basic principles of an OCSVM model [32]. Here we assume that the normal data is grouped, and the anomalies are outside of this region. Therefore, we can draw a decision boundary such that it separates the anomalies from the normal samples with as large a region as possible.

4 Problem statement

This chapter will formally introduce the new problem we propose to solve and clearly define the difference between anomalies, uninteresting anomalies, and regular data. We denote the variable space of d-dimensional features as $\mathcal{X} \subseteq \mathbb{R}^d$, x is a data sample, and x_i is the i-th dimension of a sample. Y represents the label given to the sample by the model, 0 meaning non-anomalous, and 1 meaning anomalous.

In the concrete example introduced in chapter 1, we have an anomaly detection model that analyses liver cells to detect cancerous cells. In that model, there are several features such as the size of the cell, colour, number of spikes on the outside, shape, and circumference. All these features are a combination of numerical and categorical features. There are also blood cells present, which are indeed anomalous, but we do not want to detect them.

Definition 1: Regular data
Regular data is data drawn from an underlying stationary distribution $\mathcal{D}$. The class of this regular, nonanomalous data will be referred to as NA, which stands for *nonanomalous*.

This data represents uninteresting data, where nothing out of the ordinary can be seen. This would be equivalent to the healthy liver cells in the example. All these cells look relatively the same, but this distribution can contain multiple clusters, as there are different types of liver cells. Furthermore, each cell is unique, so they all have some minor differences; however, to a domain expert, it is easy to identify them as healthy liver cells.

Definition 2: Anomaly
An anomaly is a data point that does not fit under the underlying stationary distribution $\mathcal{D}$. This class will be referred to as A, which stands for *anomalous*.

It is important to note that anomalies do represent some kind of underlying class or set of classes, and therefore, they do have an underlying distribution. However, there is no indication of what this is, and therefore no guarantee that it is disjoint from the regular data. They can also come from multiple distributions, as it is possible to have multiple types of anomalies. The underlying distribution of the class of anomalies is denoted as $\mathcal{D}''$.

In the real-world example, these are the cancerous cells. They have a distinct underlying distribution, but there can be several distributions because different types of cancerous cells can exist. Also, due to the variety of types, some types at some stages can look very similar to a healthy cell. This means that it is not given that the model can detect all anomalies.

Definition 3: Unsupervised anomaly detection
Given an unlabeled dataset X, an unsupervised model estimates the probability density function $f(x) \approx P_{\mathcal{X}\sim\mathcal{D}}(x)$, and detects an anomaly if there is a low probability that a certain sample has come from this underlying probability density function, i.e. $f(x) < \varphi$, where $\varphi > 0$ is a threshold determined either by the user of the algorithm or by the model itself.

This means that when the algorithm deems something an anomaly, the probability of it fitting under the distribution of the regular data is very low, close to 0; therefore, $f(x) < \varphi$ will return 1. This also means that when a sample is not labelled as an anomaly, it is the score of the probability is close to 1, and therefore it is finally given a label of 0.

Note that in practice, this approach can inevitably generate false positives, as we have $\varphi > 0$. In addition to this, we assume that the training data consists of only regular data, which does not contain anomalies. In reality, however, we do not have the ground truth for the data that the model is being trained on, and it may contain some anomalies. However, we assume that this percentage is very low and therefore will not affect the performance of the models.

This leads us to the final definition of non-interesting anomalies, which is defined as follows:

Definition 4: Uninteresting anomalies
An uninteresting anomaly is a data sample drawn from a distinct underlying stationary distribution D'. This class will be referred to as UA, which stands for *uninteresting anomalies*.

In our running example, these stand for the blood cells, which are very different from both the healthy and cancerous liver cells. But these do have an underlying distribution of what a blood cell looks like.

An important aspect of these uninteresting anomalies is that they have a distinct underlying distribution D', which means that we can differentiate them from the interesting anomalies. Of course, that is not always the case in reality, as some uninteresting anomalies will look too similar to the interesting anomalies and therefore be indistinguishable. Note that this distinguishability needs to be in a feature that the algorithm uses to detect the anomalies, but it can also be in multiple features or a combination of several features.

To look at how distinguishable these uninteresting anomalies are from the interesting ones, we have looked at the distance between them. For all the numerical features, we used the means to determine the distance after they were normalised. For the categorical features, we look at the ratios of each different value that the feature can take, and take the difference between the two datasets for each label, sum these up, and divide by 2 to normalise the outcome. Then, we sum up the distances of all the features and divide them by the total number of features to get a final similarity score between 0 and 1, where 1 means completely disjoint and 0 means the same. This can be summarised in the following equations:

$$\begin{aligned} d_{numeric}(A,B) &= |\hat{A} - \hat{B}| \\ d_{categorical}(A,B) &= \frac{\sum_{l \in labels} \frac{|\{a=l | a \in A\}|}{|A|} - \frac{|\{b=l | b \in B\}|}{|B|}}{2} \\ d_{total}(A,B) &= \frac{\sum_{f \in features} d(A_f, B_f)}{|features|} \end{aligned}$$

These uninteresting samples will most likely be labelled as anomalies by the unsupervised anomaly detection algorithm as they do not fit the original distribution $\mathcal{D}$. This is a correct result, as these samples are indeed anomalous, but not the desired result for the user, as these are not the samples that they are interested in. Therefore, the goal is to minimise the number of uninteresting anomalies we detect as anomalies, while still remaining accurate in detecting the real anomalies and not increasing the number of false positives we get. In mathematical terms, this means our goal is to optimise

$$\max(\alpha P(Y=0 \mid x \in NA) + \beta P(Y=1 \mid x \in A) + \gamma P(Y=0 \mid x \in UA))$$

where $\alpha + \beta + \gamma = 1$.

There are three values which can be decided by the user, which can tune the sensitivity of the model. With α, the user of the model can tune how important it is that we do not produce false negatives. This is especially important when an unnoticed anomaly is very costly, for instance, in a healthcare scenario. With β, the user can tune the impact of false positives for our use case. This would be in a case where any false positive costs a lot, for instance, if we need to shut down the servers automatically when we detect a cyber attack. Finally, with γ, the user can tune how important it is that we do not detect these uninteresting samples as anomalies.

5 Methodology

In order to no longer detect uninteresting anomalies as anomalies, the model needs to learn to detect this relatively small set of uninteresting data points as regular data as well. This is a complex task, as this framework should not be model-specific. As many of these models are unsupervised, we cannot simply tell the model that it is wrong by changing the label of a sample and adding that knowledge to the model. The solution to this is to add these data points to the training set. However, the models are designed in such a way that they are robust to some noise of anomalies in the training, as can be seen in chapter 4. Therefore, we need to find a way to make them more prominent in the dataset. This can be done by using an existing technique to artificially generate more of the uninteresting anomalies. These are then added to the dataset, and the model is retrained with this new dataset. Because of the volume of the extra data points, the models will see these data points as a proper cluster of non-anomalous data and therefore will no longer label them as an anomaly. In this chapter, this framework will be further explained in section 5.1. Then in section 5.2, we will explain how we have engineered this framework using SMOTE and what trade-offs it brought with it.

5.1. Overview of framework

There are multiple phases to the continuous retraining of the model. We start with an initial model, which is only trained with regular data in principle. Of course, some noise can exist, but research has shown that the models are generally quite resilient to this noise [33]. Then, the model is utilised in its usual manner. This means the user supplies samples to the model, or this is done automatically, depending on the context, and the anomalies are raised to the user. An important extra aspect to consider is that we make use of the explainability aspect of the model. In this process, we show the user the system's output together with an explanation of why this decision was made. This means that the model explains the decision boundaries it has found. With this information, the user can trace the reason the decision was made and see which features have been selected.

As soon as the user notices that the result is not as expected, i.e. they see that there is an anomaly that is not interesting to them, they will flag it as such. They can use the explanation that the model has given to better understand the sample and why the model chose to label this sample as such. This minimises the chance that they will label it incorrectly, which would cause the model to lose accuracy due to human error.

Based on that, a resampling phase will be triggered in which this new uninteresting anomaly is classified as normal data. This is only done once we have reached a minimum number of samples needed; for further clarification on how this number is chosen, see section 5.1.3. There are multiple possible approaches to resampling, one of which has been implemented and explained in section 5.2. After this phase, we will test the algorithm again. This is to ensure it was successful, and that this sample can now be classified correctly, as well as that the accuracy of the model on the interesting anomalies is still adequate. If the model does not meet these goals, it can try to resample again. However, this might not lead to the desired results. Instead, we are either in a situation where the class of uninteresting anomalies and that of genuine anomalies are too similar, or we do not have enough uninteresting samples gathered yet. Then the next time we encounter another uninteresting sample, we can retrain again using all gathered samples and then retrain the model in such a way that it meets our objective. An overview of this entire process can be seen in Figure 5.1.

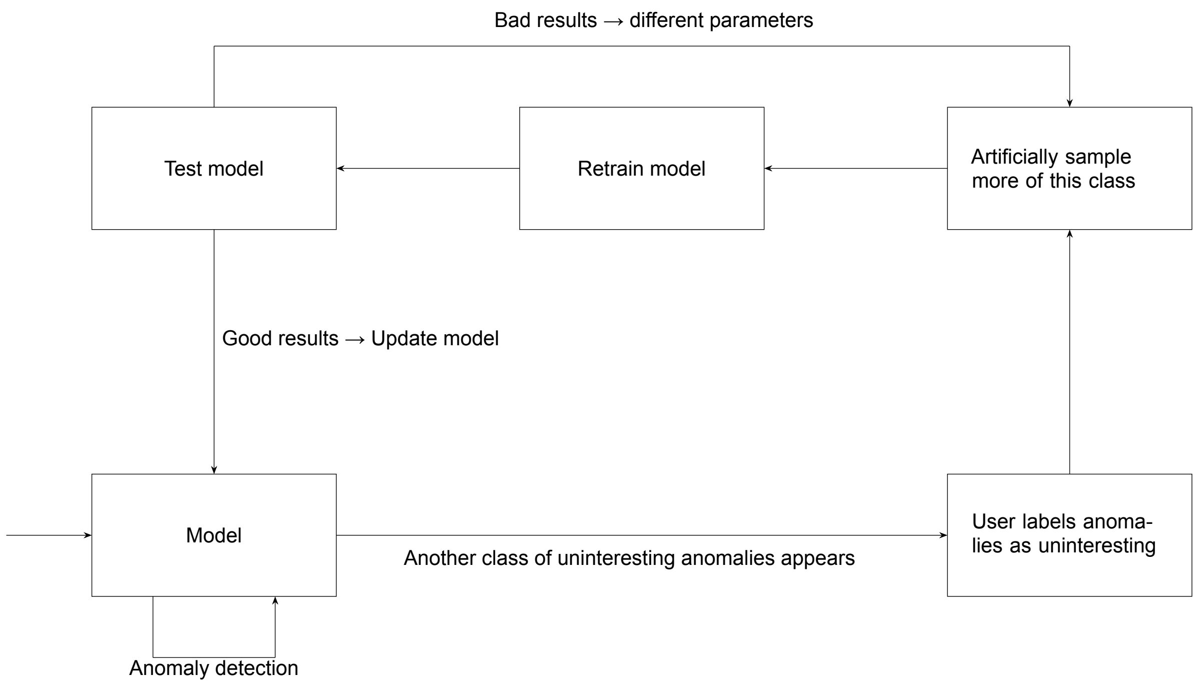

Bad results → different parameters
Test model
Retrain model
Artificially sample more of this class
Good results → Update model
Model
Another class of uninteresting anomalies appears
User labels anomalies as uninteresting
Anomaly detection

Figure 5.1: The flow of the pipeline where we retrain the model in the case of uninteresting anomalies. After retraining, the model is tested, and if it produces the desired results, the model is updated. When these results are not achieved, the resampling parameters are changed, and the model is retrained. If the class of uninteresting anomalies is deemed too similar to the interesting ones, the model is never updated, and this gets communicated to the user.

5.1.1. Guiding the user

An important aspect to consider, as mentioned in subquestion 1, is that the framework guides the user to make the right decision. In chapter 4, it was defined that the user has the ultimate say in what is an uninteresting anomaly. However, humans are not nearly as fast as computers, so they will quickly become the bottleneck [29]. Additionally, they are prone to mistakes. Therefore, we want to help the user as much as possible, such that they can perform the labelling as efficiently and accurately as possible. This was implemented by clearly showing the feature vector and highlighting the features on which the model has decided to label it an anomaly. This is especially important in cases where there are thousands of features. In addition to this, the framework displays a score of how anomalous it is, which further informs the user.

5.1.2. Testing the model

Testing this model is not as straightforward as a traditional anomaly detection algorithm, due to the presence of a third class, the uninteresting anomalies. This is also reflected in question 3, where we investigate how to test such a framework. Because of the three classes being present, we have opted to test the three accuracies apart from one another. We have done this using the true positive rate and true negative rate, see section 6.4, as well as separately testing the accuracy of the uninteresting anomalies. In addition to this, it is also important to know that the model does not get significantly worse when more samples are added. This is studied in experiment 2, see section 6.8.

5.1.3. Parameters in framework

Two parameters are important in this framework, which we will describe in the following sections.

Minimum number of samples

To prevent the model from completely deteriorating in quality as it is trying to generate many samples from too few uninteresting anomalies, we can set a minimum number needed, and only after this threshold is reached, the framework can activate. This number is, by default, set to the minimum number of samples that are needed for the resampling algorithm, which in the case of SMOTE is equal to the number of nearest neighbours it uses. For more information on what is recommended to have the minimum number of samples, see experiment 2 in section 6.8.

Critical level

It is also important to monitor the accuracy of the model and make sure it does not deteriorate. This means that domain experts need to identify the specific accuracies per class that need to be achieved by the model and what the cost is of potential false negatives. These values can be defined in many different ways, so the user can choose which they find most intuitive. For instance, they can use the overall accuracy of the system they want to ensure, or specifically, the rate of false negatives and false positives they deem to be acceptable for the system to produce.

This parameter comes to fruition in the test model state, which can be seen in figure 5.1. This is an optional parameter during the testing of the model, which checks if the system still meets the required accuracies. When these are met, the model is updated, else the procedure is tried again. If, after the second time, these desired results are not achieved, the model will not be updated. When it has not been specified, the retrained model will be updated no matter what the new accuracy achieved by the model is.

5.1.4. Possible extensions

The model that was described above is in its simplest form, but several extensions are possible that create some more freedom. These will result in a more complicated model, which is not necessary for all data sets and models. First of all, we will explain the extension with multiple types of uninteresting anomalies, which require a more complicated resampling procedure. We can also devise a model that only does intermittent retraining, which allows for a smoother user experience.

Multiple types of uninteresting anomalies

It can be a problem when there are multiple types of uninteresting anomalies. When we try to resample from a set consisting of different classes, it might not lead to any meaningful generated samples, and instead only make the algorithm lose performance. In that case, we can instead let the users label the type of uninteresting anomaly. This change also comes with some disadvantages, as it makes the job more difficult for the user, as they need to make the distinction between multiple classes. This creates a greater chance of human error. Furthermore, it also means that we have fewer available samples of all classes, which means that it will take longer until we have gathered the minimum amount of samples required for resampling. Also, the resampling method should distinguish these different classes, or it can create artificial uninteresting samples that do not look close to any of the classes, as it combines multiple classes.

Only intermittent training

Retraining can be a costly procedure due to several factors, such as the available hardware, the size of the dataset or a complex model. As a result of this, the retraining phase can become a bottleneck, as it can become a blocking factor to using the model. However, there are some solutions to this. We can choose to retrain while we are still using the old model. This minimises the amount of downtime, but on the other hand, it can be confusing for the user, which model is currently being used. In addition to this, we need to decide what needs to be done if the user labels another uninteresting anomaly; should this wait for the next retraining batch or scrap this current retraining and start over with this sample added? An alternative approach is to choose to only retrain every few samples of the uninteresting anomalies we get to minimise downtime.

5.2. Data resampling using SMOTE

To implement this framework, we have used SMOTE to resample the points such that the uninteresting anomalies make up a more significant part of the dataset. The decision to use SMOTE has been made as it is proven to be a very stable method of resampling. This is also to do with it taking into account how different features could be related to one another, as mentioned in section 3.1.1.

5.2.1. Versions of SMOTE

There are still different versions of SMOTE, even within the most basic implementation. The best fit for this use case is SMOTENC, as all of the datasets that are being investigated have a combination of numerical and categorical features, and this version supports both types. A downside of this is that it is necessary to specify beforehand which features are numerical and which ones are categorical. Luckily, this is a relatively simple step that only needs to be taken once per dataset. It is, however, something that should be taken into account while setting up the system. There are ways to automatically classify a feature as either numerical or categorical, but from manual inspection of the datasets, categorical features are sometimes encoded with a number, which leaves this approach vulnerable to miscategorisations that can lead to incorrect data resampling.

5.2.2. SMOTE-NC

SMOTE-NC stands for Synthetic Minority Over-sampling Technique for Nominal and Continuous. It is an extension of the SMOTE algorithm designed to handle datasets containing both continuous and categorical features. It operates by first identifying categorical and continuous features, where categorical features are treated as discrete variables and continuous features as numerical variables. To generate synthetic samples, SMOTE-NC employs a distance metric that combines Euclidean distance for continuous features with simple matching distance for categorical features. The algorithm identifies the k-nearest neighbours of minority class samples using this combined distance metric. For continuous features, synthetic samples are generated through interpolation, whereas for categorical features, values are randomly selected from either the original sample or its nearest neighbour. This approach allows SMOTE-NC to create realistic synthetic samples that maintain the integrity of the original dataset's relationships, making it particularly effective for addressing class imbalance in datasets with mixed feature types.

5.2.3. Hyperparameter tuning

There are two hyperparameters that need tuning in SMOTE-NC: the sampling strategy and the k-neighbours parameter. The sampling strategy encapsulates the ratio between the minority and majority class. The k-neighbours parameter is used to describe how many samples SMOTE-NC uses to generate a new sample.

A big challenge in tuning these parameters is that we need to be careful that they are not overfitted to the dataset. This can happen because we need to have uninteresting anomalies labelled to test the model, but these are not available from the start. Therefore, we have looked at both of these parameters to see if changing them gave significantly different results in the stability study. As can be seen in section 6.6, this has led to very stable results, suggesting that these parameters are not dependent on the dataset.

5.2.4. Pseudocode

In Algorithm 1, you can see how SMOTENC has been used to retrain the model. This algorithm takes as input the set of regular data and that of uninteresting anomalies and outputs the newly trained model. Initially, we need to load the datasets, which happens on lines 2 and 3 of the algorithm. Specifically, on line 3, this dataset will be the specified uninteresting anomalies by the user. After this, we calculate the original ratio and initialise SMOTE-NC. Here, we must specify which are the categorical features, which are specified when the dataset is first introduced. After this, in lines 6 and 7, we must combine the regular data and the uninteresting anomalies, as SMOTE works on a singular dataset. Here, the uninteresting anomalies have a label of 1, such that SMOTE can denote a difference in classes and resample from the minority class. After this, the actual resampling happens in line 8. Then, we need to ensure all the labels are again 0, meaning regular data, as uninteresting anomalies need to be seen as such. Finally, we can retrain the model and return it.

Algorithm 1 Oversampling using SMOTE

```
    Input: D: regular data, D': uninteresting anomalies, categorical_features
    Output: new_model

1:  procedure SMOTE-NC
2:      $x_{train}, y_{train}, \leftarrow load\_data(D)$
3:      $x_{uninteresting}, y_{uninteresting} \leftarrow load\_data(D')$
4:      $original\_ratio \leftarrow |samples_{uninteresting}|/|x_{train}| + |x_{uninteresting}|$
5:      $SMOTE \leftarrow imblearn.over\_sampling.SMOTENC(categorical\_features)$
6:      $x_{train,new} \leftarrow x_{train} \cup x_{uninteresting}$
7:      $y_{train,new} \leftarrow y_{train} \cup y_{uninteresting}$
8:      $x_{res}, y_{res} \leftarrow SMOTE.fit\_resample(x_{train,new}, x_{test,new})$
9:      $y_{res} \leftarrow np.zeros(|y_{res}|)$
10:     $new\_model \leftarrow train(x_{res}, x_{test}, y_{res}, y_{test})$
```

6 Experimental setup

Now that the methodology has been defined, it is important to test if the framework indeed produces the intended results. Therefore, in this chapter, we will describe the different experiments that have been designed to validate the framework and the role of SMOTE within it. First of all, we will describe the setup that has been used to execute these experiments, including details of the models we use in section 6.1, the baselines they are compared to in section 6.2, the datasets that are being used in section 6.3, the metrics that are considered in 6.4, and the hardware used in section 6.5. Then, we will highlight the specifics of the stability study in section 6.6, as well as the exact details of the experiments that have been run in sections 6.7, 6.8, and 6.9.

6.1. Models used

To test the framework, it was applied to the following four models:

- An Autoencoder [36];
- A Variational Autoencoder [52];
- A One-Class Support Vector Machine [3];
- An Isolation Forest [17]

We have chosen to try these out as they are all compatible with the explainable anomaly detection of [33], and they are all state-of-the-art anomaly detection models.

For the autoenconder, we have used a network of 5 layers, that encodes from the number of features to 256, 128, 64, 32 and 16. Then we use the Leaky Relu function and have a total of 50 epochs.

For the variational autoencoder, we used the same number of layers and reduction of features as in the autoencoder. Furthermore, we have 50 epochs and use the formulas as described in section 3.4.2.

For OCSVM, we use the model provided by sklearn. This model is used with the default parameters. For the nu and false positive rate parameters, we experiment with the best values by looking at the values from 0.001 to 0.03 in increments of 0.002.

For IForest, we use the sklearn model, with the number of estimators set to 500. Furthermore, we experimented with what the level of contamination gives the best score by checking all the values between 0.02 and 0.51 in increments of 0.002.

For more technical details on how these models work exactly, you can look at section 3.4.

6.2. Baselines

As this is a new kind of task, there are currently no baselines available from tools or frameworks that perform a similar task. However, there are still two baselines to which this new model is compared. The first one will be called *no extra data*, and it entails not adding any of the non-interesting data and therefore never retraining. This is just a regular anomaly detection algorithm, and as a result, we do not expect the vast majority of uninteresting anomalies to be identified as regular data. This, however, does not exclude the odd false negative that will not be detected by the model.

We also have the second baseline, in which we add the samples that have been identified as uninteresting to the training data, but we do not generate any artificial samples. This will be referred to as a *few extra*. The performance of this depends on the number of samples that are added. If there are only a few, we expect the same performance as the previous baseline, but when a lot of samples are added, we expect results similar to resampling.

6.3. Datasets used

We have used two datasets for these experiments. One of which has already been applied to anomaly detection, and as such, it has labelled the different kinds of attacks to an IOT system. These different labels are *backdoor*, *DDOS*, *DOS*, *injection*, *MITM*, *password*, *ransomware*, *scanning*, and *XSS*. Due to this, we have taken a few combinations in which we have chosen one of these types of attacks to not be interesting and therefore resample. For the initial experiments, we have put all these against each other to see the performance.

The other dataset that we used has not been applied to anomaly detection before but was instead used as an imbalanced classification set, something that happens regularly in anomaly detection [39]. For this, we chose the healthcare dataset, which is a medical dataset and therefore will also represent the case where we only consider some diagnoses to be important. This was also the dataset that is closest to the concrete example that has been introduced in chapter 1 and is freely available. In this set, there are only three labels, *normal*, *inconclusive*, and *abnormal*. Here, we have done experiments with the inconclusive class being considered the interesting anomalies class and the abnormal class being the uninteresting anomalies and vice versa.

In table 6.1, you can see how the datasets compare to one another.

Attribute	**TON-IoT**	**Healthcare dataset**
Number of Samples	1141820	49992
Number of Features	30	17
Number of Classes	10	3
Feature Type	Numerical and categorical	Numerical and categorica
Domain	Cybersecurity	Healthcare
Source	[37]	[40]

Table 6.1: Comparison of dataset characteristics.

6.4. Metrics studied

Several different metrics are used in the literature on anomaly detection. We have chosen here to consider a modified version of the true positive rate and true negative rate. This metric was also used by Li et al. [33], as well as in several other studies [23] [28].

We have, however, modified the metric to more specifically fit our use case. Instead of just having the true positive and true negative rate, it has been split into the three different classes we have defined in chapter 4. The first of these is the true positive rate, which is equivalent to the accuracy of the anomaly class:

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

Furthermore, we also have the true negative rate, which shows the accuracy of the regular data:

$$TNR = \frac{TN}{TN + FP}$$

Lastly, we also have the third metric, which we are introducing in this paper. This metric indicates how many of the uninteresting samples are picked up as regular data as opposed to anomalies, as they would originally be classified:

$$\text{Accuracy UA} = \frac{|UA| - |\{Y = 1 \mid x \in UA\}|}{|UA|}$$

Furthermore, for the final experiment, we have also looked at the time it takes to resample as well as how long the models take to retrain. This is to ensure that it will not become a bottleneck when the models are deployed.

6.5. Hardware used

These experiments were run on a MacBook M1 Pro with 32 GB of RAM. All of the experiments were run on the CPU, and the experiments where time was measured were executed 5 times and averaged to get a reliable result.

Note that this hardware is very limited, and the models are being executed on the CPU instead of a GPU. This means that when the model is applied in practice, it will have significantly faster results. This can also be seen when we compare these results to those of Li et al. [33], where they achieved significantly faster training times due to better hardware.

6.6. Stability study

As mentioned in chapter 5, we started by testing different parameters of SMOTE to ensure that this worked adequately. First of all, we looked at how the sampling strategy impacts the performance of the algorithm. For this, we have taken the number of samples of the uninteresting class to be 50. Then, we tested out the different values between 0.1 and 1 in increments of 0.1. We have used all the different models, and DDOS is considered the interesting anomaly class, and DOS is the uninteresting anomaly.

We suspect that as soon as this parameter is high enough, such that the minority class makes up a significant enough part of the model, it will achieve good results. After this, we suspect that having a higher sampling strategy to not make a significant result. However, it does impact the training time as having a lower ratio means fewer total samples.

For the second experiment, we look at how the number of nearest neighbours that the SMOTE algorithm uses to generate its samples impacts the accuracy of the model. Here the hypothesis is that around 5 neighbours is a good number, as otherwise the distance between the different samples might become too large and therefore lead to unrealistic artificial samples. This parameter is not too small either, as it is prone to continue to take the same combinations of samples.

6.7. Experiment 1: Accuracies per class

In this experiment, we have looked at the accuracies per class for all combinations of interesting and uninteresting anomalies. For this experiment, we used 50 samples randomly chosen from the uninteresting class, and the interesting class has 2950 samples. The parameters for SMOTENC in this experiment are 5 nearest neighbours and a sampling strategy of 0.5.

The hypothesis of this experiment is that if we just add the extra data without resampling, we see a small improvement in detecting uninteresting anomalies as regular data. Furthermore, when resampling is implemented, we see a significant reduction in the detection of uninteresting anomalies. However, we reason that the accuracy of the anomalies, the true positive rate, can go down. By how much this will go down depends on how distinguishable the anomalous class is from the uninteresting anomaly class. In addition to this, there is a small chance that some small clusters from the regular data will now no longer form a significant enough cluster to be detected as regular data. This can mean that the accuracy for regular data, the true negative rate, goes down, but this is not expected to be a significant amount.

6.8. Experiment 2: How many samples are needed

One of the most important aspects to consider in our framework is how many samples are needed to lead to meaningful results. This is especially important because this problem may suffer from a cold start, where at the beginning there are so few samples of the uninteresting class that the model simply cannot use them for resampling yet, as this can be detrimental to the performance of the system. Therefore, we have looked into how many samples are needed to get accurate results. Note that the minimum number of samples required by the model is equal to the number of nearest neighbours SMOTE uses plus 1, which is 6 in our case.

The hypothesis of this experiment is more difficult to make as it is dependent on both the uninteresting and interesting anomaly classes to determine how many samples are needed. If the uninteresting anomalies is a very shallow class, meaning that there is very little variance between the different samples, it will not need many samples to achieve good results. However, when the uninteresting class consists of multiple clusters, it will need more. As a baseline we expect that with around 50 samples the model can achieve good results.

We also note that when too many samples of the uninteresting class are supplied, the scenario may arise that the problem gets flipped, and the regular class might become the minority. However, since SMOTE calculates what the minority class is, it will automatically flip it and not lead to regular data being detected as anomalous. Furthermore, when the framework is working as intended after the user has labelled sufficient uninteresting classes, SMOTE will be able to model the uninteresting class sufficiently. This means the user should not encounter and therefore label many more uninteresting samples, making it nearly impossible to overshadow the regular data class.

6.9. Experiment 3: Training time

An important aspect to consider is how long resampling and retraining will take, as we want to ensure that this process does not become the bottleneck. This is especially important, as with this new framework, the model is being retrained many times as opposed to the static model in traditional anomaly detection. Therefore, we have timed how long these different methods take. Here, we used 50 uninteresting samples and took a resampling rate of 0.5. The dataset originally contained 3000 samples. This means that the new dataset will consist of 4500 samples. Note that the training times will scale depending on the number of samples that are given in both the interesting and uninteresting classes and how many dimensions these samples have.

It is expected that the time it takes for the algorithm to run will go up. This is because of two main reasons. First of all, it is being supplied with more data and therefore will take longer to go through all the samples for each epoch. But in addition to this, we are also now supplying it with data with a more difficult distribution. This means the model might need more epochs to find the optimal function that can detect the anomalies.

7 Results

In the following chapter, we will highlight the results that our framework has achieved when executing the experiments described in chapter 6. Here we found that the framework successfully reduces false positives by learning to ignore uninteresting anomalies, especially when these are sufficiently distinct from real anomalies. SMOTE proves effective and stable under most settings, with around 50 samples generally being sufficient for reliable performance. Although resampling increases training time by around 60%, it significantly improves model accuracy and generalisability across different datasets and models.

In section 7.1, we will discuss the results achieved in the stability study. After this, we discuss the overall accuracies that have been achieved in 7.2. After this, we ran the experiments to see more specifically how many samples are required in 7.3. In addition to this, we also see how the times of retraining have changed due to this framework in section 7.4. Finally, in section 7.5, we discuss how generalisable this model is to other datasets.

7.1. Stability study

We have found that the model is relatively stable under the different parameters of SMOTE. As long as we add enough new samples, such that the ratio is at least 0.2, the model can detect the uninteresting anomalies, and the differences in accuracy of all the different classes are very close to one another. This is also reflected in figure 7.1, where we ran the experiment on the IForest model. To verify this is also the case for the other models, other experiments for the other models have been run and can be seen in appendix A. In these figures, it can be seen that these are even less dependent on the learning rate, which lines up with the way these models function. Additionally, we found that performing more resampling does not worsen the performance in any of the classes. However, with a higher ratio, the models take longer to retrain, as they now have more samples to reconsider for every epoch. Therefore, for the rest of the experiments, we have decided to keep this value at 0.5. All of these findings were in line with our hypothesis.

We have found that the number of neighbours is a more sensitive parameter, where anything over 5 can lead to very volatile results, as can be seen in figure 7.2. Sometimes, the accuracy of the uninteresting anomalies is still excellent, but it can also drop very drastically. This is because when taking too many samples to create a new sample, there might not be enough nearby data points to sample from, which makes the resulting new artificial data point not representative enough of the uninteresting anomalies. For the graphs of the results of the other models, see appendix A.

To conclude, we can see that SMOTE performs relatively similarly, no matter the parameters we use, as long as we stay in a safe zone. We were able to reproduce these results using a different model and dataset. This means that when we use this model on a different dataset, there is no need to figure out the SMOTE parameters again. This is especially useful in cases where we do not know beforehand what the uninteresting anomalies will be.

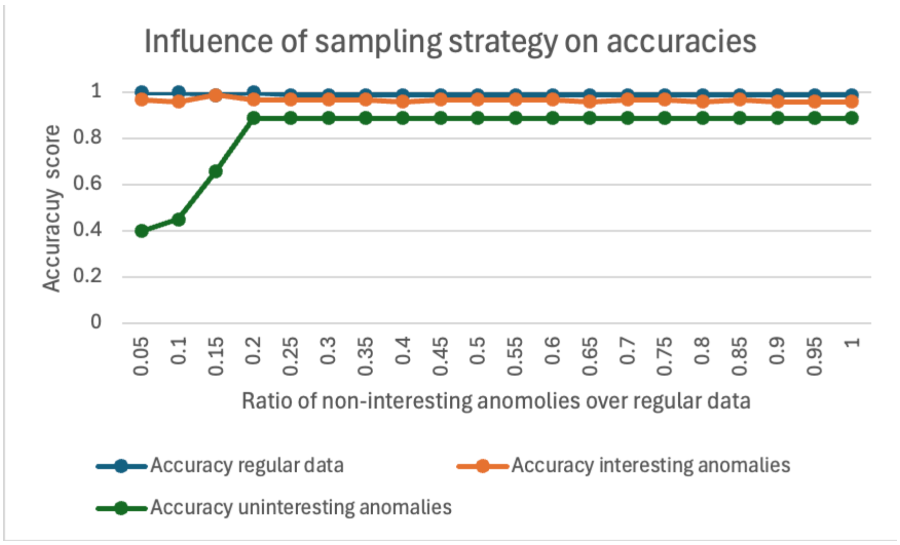


Figure 7.1: The influence of different ratios of uninteresting anomalies to regular data on the accuracies of all the classes. This experiment has been run on the IForest model. Here it can be seen that the accuracy for regular data and that of anomalies is very stable and close to 1. The accuracy of the uninteresting anomalies is lower at first, but is steady from a ratio of 0.2 onwards.

7.2. Experiment 1: Accuracies per class

From the first experiment, it can be seen that resampling indeed improves the framework such that it no longer detects uninteresting anomalies as anomalies. This is a promising result, as we can see that without resampling, adding uninteresting anomalies to the training set does help a little bit, but it is not the solution. There is, however, a small drop in the rate at which real anomalies are picked up, as there is a small chance that an anomaly is labelled as uninteresting, as more samples are now considered uninteresting. It can be seen that this applies to both datasets in tables 7.1 and 7.2.

However, we can see that the algorithm's performance heavily depends on which anomalies are interesting and uninteresting we give it. This is especially apparent in the results that can be seen in appendix B, where some combinations of interesting and uninteresting anomalies do not lead to meaningful results in any of the models. We can see that a crucial factor in the algorithm's performance is that there is a significant difference between the distribution of a regular anomaly and an uninteresting one. To investigate this further, we looked deeper into the differences between the classes based on the similarity score of the averages of the two classes. The similarity scores can be seen in table 7.3, where when we cross-reference the accuracies with this table, the more similar the two classes are, the worse the results are. Whenever they are very similar to one another, it can be seen in one of two ways. Either the model becomes very good at no longer detecting the uninteresting anomalies, but loses all accuracy in the anomalies class, or vice versa.

Furthermore, we found that the performance is different for each model. Here we can see that some models are better suited for a more complex underlying distribution. This is because the new distribution will have multiple clusters, some of which are uninteresting. We found that the AE and VAE models generally tend to obtain the best results, where we see the smallest drop in performance of the anomalies and regular data. Here we see that IForest leads to very unstable results, and the relatively simple model of OCSVM struggles to encapsulate this more complicated distribution.

Finally, we found that we need to ensure that the data samples we give are representative of the uninteresting samples. We found that for some of the combinations, resampling has not brought about a dramatic drop in performance for anomalies. Here, the algorithm struggles to pick up the overall trend of the distribution of interesting anomalies and therefore cannot lead to a sufficient result. We also see that it is often not possible to get 100% of the uninteresting samples to be labelled as such; some of them will continue slipping through. This is because we are generalising what the data looks like, and there will always be anomalies that look different from the majority and therefore cannot be caught by the model.

Model	Which method	Accuracy Anomalies	Accuracy regular data	Accuracy uninteresting anomolies
IForest	No extra data	1	0.9705	0
IForest	Few extra	1	0.9838	0
IForest	Resampling SMOTE	0.9991	0.9743	0.8901
AE	No extra data	1	0.9981	0
AE	Few extra	0.9977	00.9705	0.5132
AE	Resampling SMOTE	0.9978	0.9619	1
OCSVM	No extra data	1	0.9790	0
OCSVM	Few extra	0.9977	0.9810	0.6394
OCSVM	Resampling SMOTE	0.4210	0.9820	0.9235
VAE	No extra data	1	0.9980	0
VAE	Few extra	0.9977	0.9609	0.4758
VAE	**Resampling SMOTE**	**0.9979**	**0.9458**	**0.9926**

Table 7.1: Comparison of how different models perform on the same combination of interesting anomaly and uninteresting anomaly. All of these experiments have been run on *backdoor* being considered the interesting anomaly and *scanning* the uninteresting anomaly As can be seen from these results, adding just a few extra of the samples already achieves some results, although it depends on the model how sensitive the model is for this when resampling using SMOTE is added, the results are excellent. We do note that there are always anomalies in the anomalies class, and we cannot perfectly capture the distribution of the uninteresting anomaly, which ensures that the accuracy never goes up to 1. Highlighted can be seen as the best combination of all three factors.

Model	Which method	Accuracy Anomalies	Accuracy regular data	Accuracy uninteresting anomolies
IForest	No extra data	**0.8536**	**0.9541**	0
IForest	Few extra	0.8536	0.9320	0.2431
IForest	Resampling SMOTE	0.8513	0.9320	**0.8432**
AE	No extra data	**0.8414**	**0.9541**	0.0045
AE	Few extra	0.8354	0.9267	0.2457
AE	Resampling SMOTE	0.8277	0.9110	**0.8615**
OCSVM	No extra data	**0.8612**	**0.9768**	0
OCSVM	Few extra	0.8124	0.9541	0.2410
OCSVM	Resampling SMOTE	0.5142	0.9441	**0.8324**
VAE	No extra data	**0.8458**	**0.9325**	0.0512
VAE	Few extra	0.8458	0.9271	0.5413
VAE	Resampling SMOTE	0.8456	0.9254	**0.9521**

Table 7.2: The performance of the different models using our framework on the healthcare dataset, where abnormal is the uninteresting anomaly and inconclusive interesting. As can be seen from the results, all of the different models perform relatively similarly on the baseline of no extra data. There they also achieve the best results for the accuracy of anomalies and that of regular data. However, resampling using SMOTE leads to the best results for uninteresting anomalies by a significant margin. Highlighted are per model which method gave the best accuracy per class.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		0.2584	0.8536	0.2466	0.1988	0.9351	0.7659	0.3699	0.3288
DDOS	0.2584		0.8337	0.0156	0.1746	0.4368	0.5123	0.8577	0.4655
DOS	0.8536	0.8337		0.2568	0.4587	0.3212	0.5762	0.6542	0.7548
Injection	0.2466	0.0156	0.2568		0.2155	0.3459	0.4366	0.6066	0.4544
MITM	0.1988	0.1746	0.4587	0.2155		0.4489	0.3481	0.8932	0.3574
Password	0.9351	0.4368	0.3212	0.3459	0.0489		0.4954	0.7952	0.3577
Ransomware	0.7659	0.5123	0.8762	0.4366	0.3481	0.4594		0.8802	0.2147
Scanning	0.3699	0.8577	0.6542	0.6066	0.8932	0.7942	0.8802		0.3562
XSS	0.3288	0.4655	0.7548	0.4544	0.3574	0.3577	0.2147	0.3562	

Table 7.3: Similarity scores of the different types of anomalies against each other. These scores have been calculated per the definition in chapter 4.

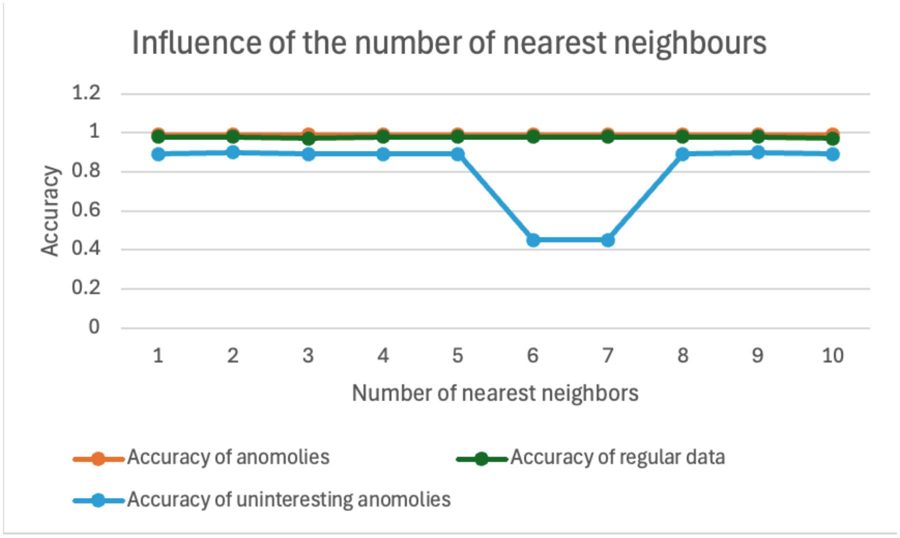


Figure 7.2: The influence of the number of nearest neighbours the SMOTE algorithm uses on the accuracies of all the different classes using IForest. Here it can be seen any number below 5 has very stable results, after which it becomes unstable.

7.3. Experiment 2: How many samples are needed

With this experiment, we found that the number of samples that are needed to differentiate each type of uninteresting anomaly varies greatly. However, we also found that the results never deteriorated by taking more samples, so it is always beneficial to gather more data.

We found that it heavily depends on the model and how many samples we need to gather to achieve results. We found that, in general, around 50 samples were sufficient for the majority of combinations of interesting and uninteresting anomalies to be able to effectively distinguish them. This is still a high number of samples to gather, and depending on the number of uninteresting anomalies in the dataset, this can take a while. However, if there are not as many uninteresting anomalies in the dataset, this also means that they do not significantly inhibit the performance of the algorithm. This can also be seen in figure 7.3. This graph also shows that it is still a very unstable process and you need to ensure that to have enough samples to ensure reliable results. In appendix C, we can see how different combinations of interesting anomalies, uninteresting anomalies and models lead to diverse results. Some of the combinations can have a cold start, which is in line with our hypothesis.

7.4. Experiment 3: Training time

As can be seen in table 7.4, the runtime of resampling is on average around 60% more compared to no resampling. From the experiments, we see that the resampling itself only takes milliseconds, and therefore, the time it takes is negligible when considering the performance of the framework. However, it is also clear that the time it takes to run the models significantly increases, as more time is needed to go through all the samples in each epoch. In addition to this, more epochs might be needed depending on the system as the data becomes more difficult to distinguish from one another. This is something we need to be careful with because, with very big datasets or datasets with many features, this will become a bottleneck to the system. There are, however, some solutions to this problem that can help mitigate it. These solutions are discussed in section 5.1.4, where intermittent training is mentioned.

7.5. Generalisability to other datasets

As can be seen from the results of experiment 1, this new model does not always work as intended, because uninteresting and interesting anomalies can look too similar when you look at the data. However, as we have attained good results with two different data sets from completely different domains in anomaly detection, as well as for different types of state-of-the-art models, we believe that this framework is very generalisable to other datasets.

However, we need to ensure that our classes are well-distinguishable from each other. When the un-

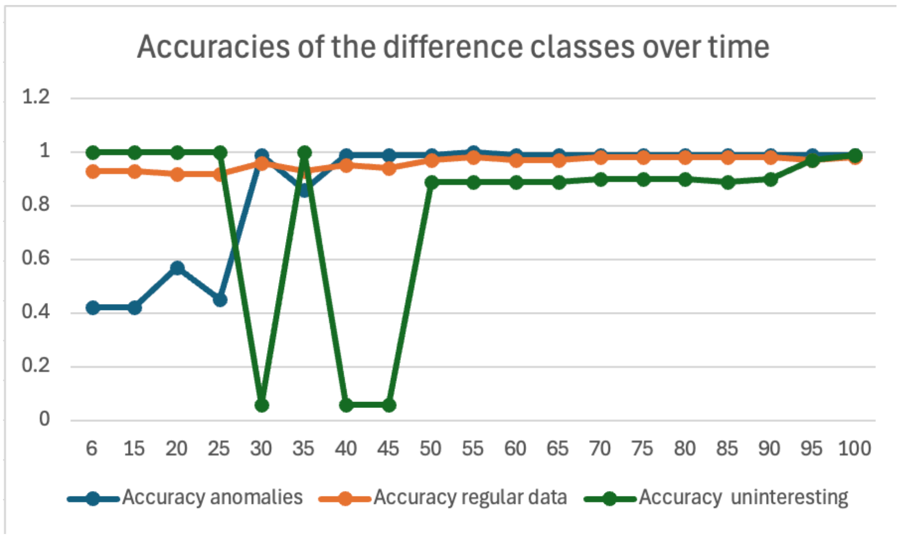


Figure 7.3: The number of samples needed to create reliable results for IForest. Here we can see that initially the model is very unreliable, but after 50 samples are gathered, we can achieve good results. We also see that as the number of samples increases, we can achieve even better results, as can be seen around 95 samples collected.

		Time (seconds)
OCSVM	No resampling	5.368
OCSVM	Resampling	8.648
AE	No resampling	7.369
AE	Resampling	11.456
IForest	No resampling	15.348
IForest	Resampling	24.158
VAE	No resampling	4.428
VAE	Resampling	7.145

Table 7.4: The impact of resampling on the runtime of the training, with a learning rate of 0.5 and 50 unknown samples. These values have been averaged over 5 runs. From these results, we see that resampling consistently adds approximately 60% to the runtime as opposed to that without resampling.

interesting anomalies look too similar to the anomalies that we are interested in, we simply cannot guarantee correct results. We can guarantee a degree of performance by calculating the distance metric as defined in chapter 4 and ensuring that it is higher than 0.5, as we found that for any combination of uninteresting and interesting classes that exceeds this, we always acquire good results with an appropriate model.

8 Future work

As can be seen from chapter 7, the framework we designed has shown promising results when the classes are sufficiently distinct. However, several limitations have been found, too. Therefore, in this chapter, potential extensions to this algorithm will be proposed. In section 8.1, we will explain how to extend the framework to handle non-numeric data. Following this, in section 8.2, we will propose conducting more comprehensive user testing with domain experts. Furthermore, section 8.3 describes an additional step that can be added to the framework, where it calculates beforehand if using our proposed technique is feasible. We discuss how heuristics can be used in this method in section 8.4. Finally, in section 8.5 we discuss how the creation of offspring in genetic algorithms can be used to fill in the resampling stage in the framework.

8.1. Non-numeric data

The method we have chosen for resampling is designed exclusively for numerical and categorical data. It relies on data points and statistical resampling methods. However, many anomaly detection problems involve images. Although we can still incorporate images by collecting metadata that summarises them, directly inputting images into this framework is not currently feasible. For future work, we can enhance this framework to support images as well. In theory, the current technique allows for pixel values, albeit with slight modifications; however, averaging a few images may not lead to new samples that make any sense, as the specific pixel values can be very different but still represent the same underlying class. Therefore, we need to look into other techniques for image data. One possible approach is to employ generative AI models to create new samples and use these to train the model, but this is computationally expensive. Another possibility is to splice multiple pictures together, which is closer to what SMOTE does when combining multiple samples.

8.2. Testing with domain experts

Due to time constraints, we were unable to test the framework with its primary users to evaluate its effectiveness. Therefore, as part of future work, there should be a detailed study conducted to assess the framework in use, preferably using data from previously unconsidered use cases. This study would include benchmarks that provide valuable insights into how to create the most realistic artificial samples of the uninteresting class. This is done by presenting users with newly generated samples and asking whether these samples align with their expectations. By gathering user feedback and evaluating the framework's performance in real-world scenarios, we can identify areas for improvement and refine the system to meet user needs more effectively.

8.3. Stastical testing

As can be seen from the results in section 7.2, it is not always possible to distinguish an uninteresting class from the actual anomalies, as the possibility of this heavily depends on how distinct these two classes are from one another. The distance between the averages of the two classes is taken as a measure of how well-suited this model is to the elimination of the uninteresting class. This test is currently not included in the framework, but it can benefit from having this test integrated. In this test, which will be introduced as a new stage before the framework starts artificially resampling, the distance will be checked against a threshold to ensure that it is possible to attain substantial results. This threshold can be set as a parameter. The retraining phase is not triggered if the distance is too small. We can also extend these statistical tests to capture more information about the data, such as the standard deviation. In addition to this, we can enhance this metric even further by only taking into account the features that are actually used to make the decision, as dictated by the explainability algorithm.

8.4. Inclusion of heuristics in custom method

Another possible improvement is to tailor a special method that does the resampling instead of using SMOTE. This method can then be specifically designed for this task, making it more suitable. This also means that the framework can take heuristics into account. This allows us to be more in control of which samples it uses to resample; for instance, we can make sure it uses samples that are spread apart. Outside of that, we can add additional heuristics which look at the difference in the mean of a feature between the uninteresting anomalies and regular data, and if those means are close enough, we can resample using the regular data as well.

8.5. Genetic algorithm techniques

An additional idea is to use genetic algorithm techniques to resample the data. In that domain, many ways of making new offspring of a few samples have been developed, for features that are continuous, integer, categorical, and binary. It would be interesting to experiment with using these techniques for resampling, where the different features are genes, and then we use a genetic algorithm to make new offspring to obtain generated samples.

9 Conclusion

Anomaly detection plays a crucial role in identifying irregular patterns in data across various domains, from cybersecurity to healthcare. However, a persistent challenge in anomaly detection systems is their inability to distinguish between truly relevant anomalies and those that, while statistically anomalous, are irrelevant to the user. To address this issue, this thesis introduced a framework that retrains anomaly detection models to suppress these uninteresting anomalies by leveraging user feedback and synthetic data resampling.

The findings of this research demonstrate that by incorporating the Synthetic Minority Over-sampling Technique for Nominal and Continuous (SMOTE-NC), we can augment the training dataset with artificial samples of uninteresting anomalies, thereby teaching the model to recognise them as regular data. This approach was tested using four state-of-the-art anomaly detection models: Isolation Forest, One-Class Support Vector Machines, Autoencoders, and Variational Autoencoders. This was then tested across two datasets in cybersecurity and healthcare, respectively. The results indicate that when interesting and uninteresting anomalies are sufficiently distinct, the framework significantly reduces false positives while only minimally compromising the model's ability to detect relevant anomalies. This is also dependent on the model which is used, with some showing significantly better results. In cases where the two anomaly classes were highly similar, the impact of the framework was less pronounced, highlighting the importance of clear feature differentiation. In these situations, the accuracy of the interesting anomalies class is significantly reduced. Furthermore, the framework needs quite a few samples, around 50, to lead to meaningful results. This is, however, not a big obstacle, because this framework is meant to function when we repeatedly get false positives from the same uninteresting class.

The key takeaways from this research are threefold. First, user feedback is an essential component in refining anomaly detection models, ensuring that flagged anomalies align with domain expertise. Second, data augmentation techniques like SMOTE-NC can effectively reshape anomaly detection behaviour, provided that the uninteresting anomalies have distinguishable characteristics. Lastly, the success of this framework depends on the degree of separability between interesting and uninteresting anomalies, suggesting that future work should explore adaptive feature selection methods to enhance distinction.

In conclusion, this thesis contributes a novel and practical approach to improving anomaly detection systems by mitigating the impact of uninteresting anomalies. By integrating domain knowledge and resampling techniques, the proposed framework provides a scalable solution for dynamic anomaly detection tasks. Future research could focus on optimising retraining strategies, exploring real-time adaptation, and extending this approach to other domains where anomaly detection remains a critical challenge.

References

[1] Mostofa Ahsan, Rahul Gomes, and Anne Denton. “Smote implementation on phishing data to enhance cybersecurity”. In: *2018 IEEE international conference on electro/information technology (EIT)*. IEEE. 2018, pp. 0531–0536.

[2] Bálint Bicski, Adrián Pekár, and Károly Farkas. “Anomaly Detection in Industry 4.0 using Probabilistic Approaches”. In: ().

[3] Adel Binbusayyis and Thavavel Vaiyapuri. “Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class SVM”. In: *Applied Intelligence* 51.10 (2021), pp. 7094–7108.

[4] Rafał Brociek et al. “Application of heuristic algorithms in the tomography problem for pre-mining anomaly detection in coal seams”. In: *Sensors* 22.19 (2022), p. 7297.

[5] Andy Brown et al. “Recurrent neural network attention mechanisms for interpretable system log anomaly detection”. In: *Proceedings of the first workshop on machine learning for computing systems*. 2018, pp. 1–8.

[6] Weipeng Cao et al. “Research progress of zero-shot learning beyond computer vision”. In: *Algorithms and Architectures for Parallel Processing: 20th International Conference, ICA3PP 2020, New York City, NY, USA, October 2–4, 2020, Proceedings, Part II 20*. Springer. 2020, pp. 538–551.

[7] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. “Anomaly detection using one-class neural networks”. In: *arXiv preprint arXiv:1802.06360* (2018).

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[9] Ming-Wei Chang et al. “Importance of Semantic Representation: Dataless Classification.” In: *Aaai*. Vol. 2. 2008, pp. 830–835.

[10] Nitesh V Chawla et al. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[11] Nwodo Benita Chikodili et al. “Outlier detection in multivariate time series data using a fusion of K-medoid, standardized euclidean distance and Z-score”. In: *International Conference on Information and Communication Technology and Applications*. Springer. 2020, pp. 259–271.

[12] Ashesh Das. *Oversampling to remove class imbalance using SMOTE*. `https://medium.com/@asheshdas.ds/oversampling-to-remove-class-imbalance-using-smote-94d5648e7d35`. [Accessed 16-04-2025].

[13] Pandit Byomakesha Dash et al. “Model based iot security framework using multiclass adaptive boosting with smote”. In: *Security and Privacy* 3.5 (2020), e112.

[14] *Dictionary - Cancer: Liver cancer - The Human Protein Atlas*. `https://v16.proteinatlas.org/learn/dictionary/cancer/liver+cancer/detail+1/magnification+1`. [Accessed 14-04-2025].

[15] C Ding et al. “Cluster consistency: simple yet effect robust learning algorithm on large-scale photoplethysmography for atrial fibrillation detection in the presence of real-world label noise”. In: *arXiv preprint arXiv* 2211 (2022).

[16] Zhiguo Ding and Minrui Fei. “An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window”. In: *IFAC Proceedings Volumes* 46.20 (2013), pp. 12–17.

[17] Yutao Dong et al. “{HorusEye}: A Realtime {IoT} Malicious Traffic Detection Framework using Programmable Switches”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 571–588.

[18] Min Du et al. “Lifelong anomaly detection through unlearning”. In: *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019, pp. 1283–1297.

[19] Alberto Fernández et al. “SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary”. In: *Journal of artificial intelligence research* 61 (2018), pp. 863–905.

[20] Tharindu Fernando et al. “Deep learning for medical anomaly detection–a survey”. In: *ACM Computing Surveys (CSUR)* 54.7 (2021), pp. 1–37.

[21] Erik M Ferragut, Jason Laska, and Robert A Bridges. “A new, principled approach to anomaly detection”. In: *2012 11th International Conference on Machine Learning and Applications*. Vol. 2. IEEE. 2012, pp. 210–215.

[22] Leilani H Gilpin et al. “Explaining explanations: An overview of interpretability of machine learning”. In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE. 2018, pp. 80–89.

[23] Fabio A González and Dipankar Dasgupta. “Anomaly detection using real-valued negative selection”. In: *Genetic Programming and Evolvable Machines* 4 (2003), pp. 383–403.

[24] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. “Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning”. In: *International conference on intelligent computing*. Springer. 2005, pp. 878–887.

[25] Junho Hong, Chen-Ching Liu, and Manimaran Govindarasu. “Integrated anomaly detection for cyber security of the substations”. In: *IEEE Transactions on Smart Grid* 5.4 (2014), pp. 1643–1653.

[26] Yasuhiro Ikeda et al. “Human-assisted online anomaly detection with normal outlier retraining”. In: *Proc. ACM SIGKDD Workshop*. 2018.

[27] Matloob Khushi et al. “A comparative performance analysis of data resampling methods on imbalance medical data”. In: *IEEE Access* 9 (2021), pp. 109960–109975.

[28] Yuma Koizumi et al. “SNIPER: Few-shot learning for anomaly detection to minimize false-negative rate with ensured true-positive rate”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 915–919.

[29] Konstantin Kueffner et al. “Into the unknown: active monitoring of neural networks (extended version)”. In: *International Journal on Software Tools for Technology Transfer* 25.4 (2023), pp. 575–592.

[30] Donghwoon Kwon et al. “An empirical study on network anomaly detection using convolutional neural networks”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1595–1598.

[31] *Leukemia &amp; Blood Chemistry*. `https://serc.carleton.edu/woburn/resources/leukemia_images.html`. [Accessed 14-04-2025].

[32] Chenyang Li et al. “Lifelong condition monitoring based on NB-IoT for anomaly detection of machinery equipment”. In: *Procedia Manufacturing* 49 (2020), pp. 144–149.

[33] Ruoyu Li et al. “Interpreting Unsupervised Anomaly Detection in Security via Rule Extraction”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[34] Zhong Li, Yuxuan Zhu, and Matthijs Van Leeuwen. “A survey on explainable anomaly detection”. In: *ACM Transactions on Knowledge Discovery from Data* 18.1 (2023), pp. 1–54.

[35] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation-Based Anomaly Detection”. In: 6.1 (Mar. 2012). issn: 1556-4681. doi: `10.1145/2133360.2133363`. url: `https://doi.org/10.1145/2133360.2133363`.

[36] Yisroel Mirsky et al. “Kitsune: an ensemble of autoencoders for online network intrusion detection”. In: *arXiv preprint arXiv:1802.09089* (2018).

[37] Nour Moustafa. *ToN*$_{IoT datasets}$. 2019. doi: `10.21227/fesz-dm97`. url: `https://dx.doi.org/10.21227/fesz-dm97`.

[38] David FN Oliveira et al. "A new interpretable unsupervised anomaly detection method based on residual explanation". In: *IEEE Access* 10 (2021), pp. 1401–1409.

[39] Guansong Pang et al. "Deep learning for anomaly detection: A review". In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–38.

[40] Prasad Patil. *Healthcare Dataset — kaggle.com*. `https://www.kaggle.com/datasets/prasad22/healthcare-dataset/data`. [Accessed 17-04-2025].

[41] Dan Pelleg and Andrew Moore. "Active learning for anomaly and rare-category detection". In: *Advances in neural information processing systems* 17 (2004).

[42] Jia Pengfei, Zhang Chunkai, and He Zhenyu. "A new sampling approach for classification of imbalanced data sets with high density". In: *2014 international conference on big data and smart computing (BIGCOMP)*. IEEE. 2014, pp. 217–222.

[43] Marco AF Pimentel et al. "A review of novelty detection". In: *Signal processing* 99 (2014), pp. 215–249.

[44] Yousra Regaya, Fodil Fadli, and Abbes Amira. "Point-Denoise: Unsupervised outlier detection for 3D point clouds enhancement". In: *Multimedia Tools and Applications* 80.18 (2021), pp. 28161–28177.

[45] Bernardino Romera-Paredes and Philip Torr. "An embarrassingly simple approach to zero-shot learning". In: *International conference on machine learning*. PMLR. 2015, pp. 2152–2161.

[46] Stefania Russo et al. "Active learning for anomaly detection in environmental data". In: *Environmental Modelling & Software* 134 (2020), p. 104869.

[47] Burr Settles. "Active learning literature survey". In: (2009).

[48] Rushikesh Shende. *Autoencoders, Variational Autoencoders (VAE) and β-VAE*. `https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d`. [Accessed 16-04-2025].

[49] Andrea Stocco and Paolo Tonella. "Towards anomaly detectors that learn continuously". In: *2020 IEEE international symposium on software reliability engineering workshops (ISSREW)*. IEEE. 2020, pp. 201–208.

[50] Feiyu Xu et al. "Explainable AI: A brief survey on history, research areas, approaches and challenges". In: *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*. Springer. 2019, pp. 563–574.

[51] Hongzuo Xu et al. "Deep isolation forest for anomaly detection". In: *IEEE Transactions on Knowledge and Data Engineering* (2023).

[52] Xing Xu et al. "Toward effective intrusion detection using log-cosh conditional variational autoencoder". In: *IEEE Internet of Things Journal* 8.8 (2020), pp. 6187–6196.

[53] Abdulmalik Shehu Yaro et al. "Outlier Detection Performance of a Modified Z-Score Method in Time-Series RSS Observation with Hybrid Scale Estimators". In: *IEEE Access* (2024).

[54] Daochen Zha et al. "Meta-AAD: Active anomaly detection with deep reinforcement learning". In: *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2020, pp. 771–780.

[55] Zeng-Guang Zhou and Ping Tang. "Continuous anomaly detection in satellite image time series based on z-scores of season-trend model residuals". In: *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE. 2016, pp. 3410–3413.

A

Stability study results

In the following figures, the rest of the results of the stability study can be seen.

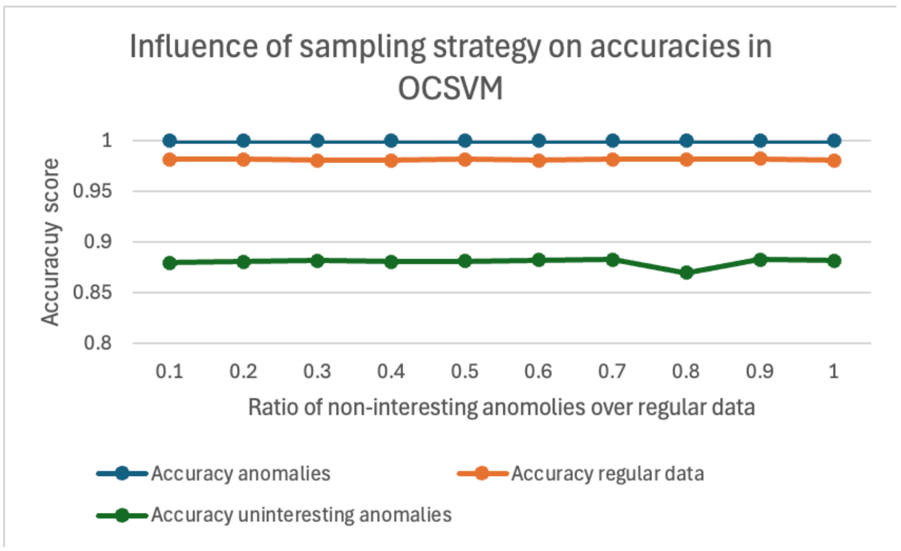


Figure A.1: The influence of different ratios of uninteresting anomalies to regular data on accuracies of all the classes. This experiment has been run on the OCSVM model. Here it can be seen that this model gives very stable results.

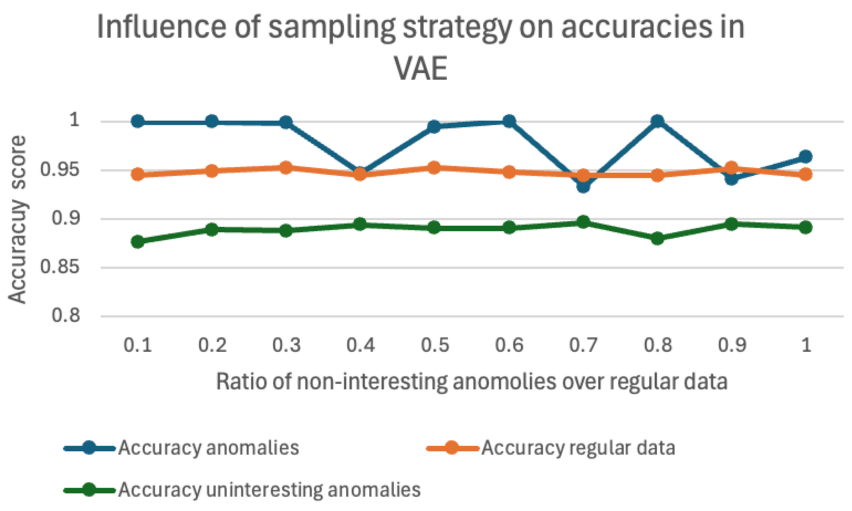


Figure A.2: The influence of different ratios of uninteresting anomalies to regular data on accuracies of all the classes. This experiment has been run on the VAE model. Here it can be seen that this model gives slightly more unstable results, due to the randomness that is in the algorithm. It can be concluded that there is no obvious learning rate where the performance is superior.

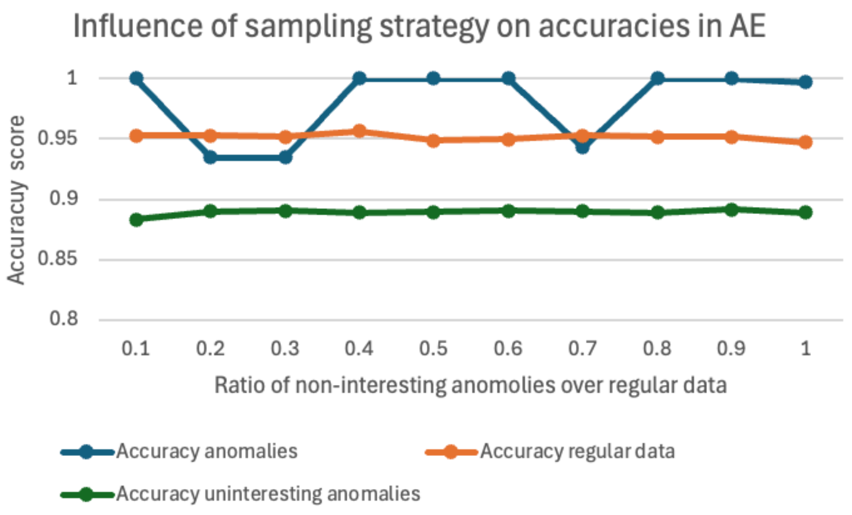


Figure A.3: The influence of different ratios of uninteresting anomalies to regular data on accuracies of all the classes. This experiment has been run on the AE model. Here it can be seen that this model gives slightly more unstable results, due to the randomness that is in the algorithm. It can be concluded that there is no obvious learning rate where the performance is superior.

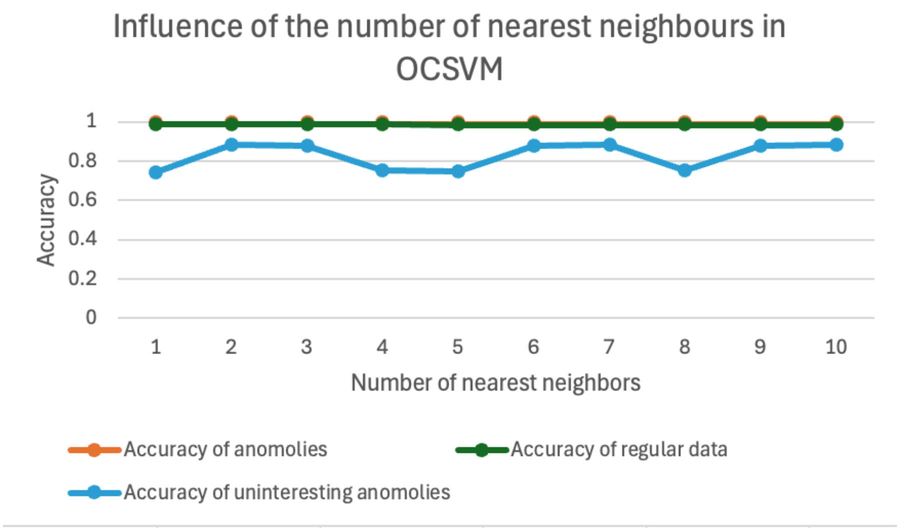


Figure A.4: The influence of the number of neighbours on the accuracies of all the classes. This experiment has been run on the OCSVM model. Here it can be seen that this model gives quite stable results, and there is not really a pattern to when it is better or worse. This suggests it is up to the randomness in SMOTE that makes a difference in the results.

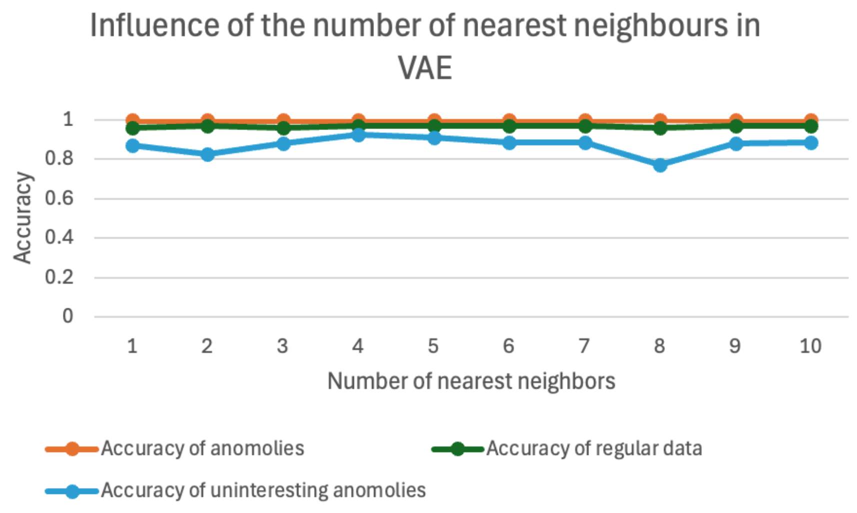


Figure A.5: The influence of the number of neighbours on the accuracies of all the classes. This experiment has been run on the VAE model. Here it can be seen that this model gives quite stable results, only with a higher number of neighbours producing slightly worse results.

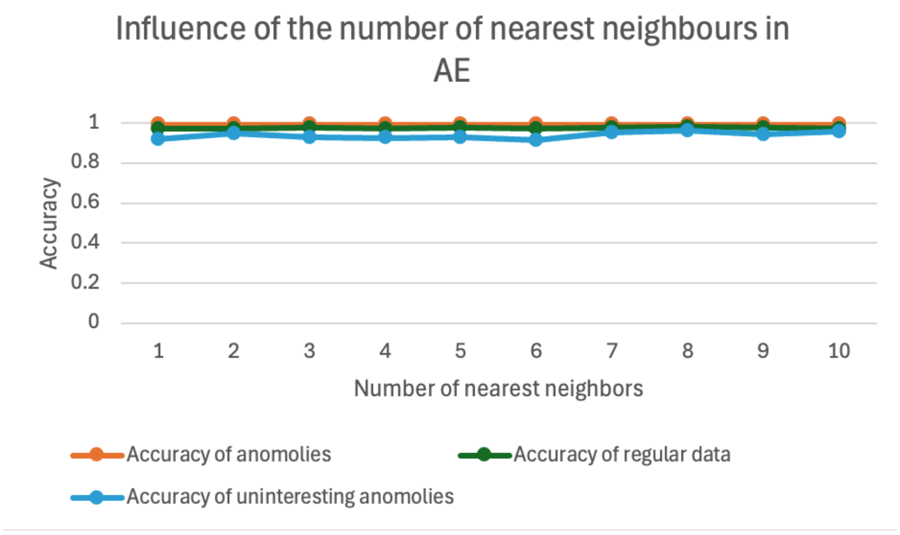


Figure A.6: The influence of the number of neighbours on the accuracies of all the classes. This experiment has been run on the AE model. Here, it can be seen that this model gives very stable results.

B Results experiment 1

Here we have given the results for all the different experiments we have ran for experiment 1. Here the anomaly listed at the top is defined as the interesting anomaly, and the one to the side is the uninteresting one. Then inside each cell, we first list the true positive rate, which entails the accuracy over the anomaly class, then the true negative rate, and finally the rate of accuracy of an uninteresting anomaly being labelled as regular data.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	0.99986 0.98136 0.40343	0.99787 0.98399 0.09516	0.99906 0.98881 0.08590	1.0 0.98638 0.40783	0.99997 0.98835 0.39183	0.99828 0.98276 0.35445	1.0 0.97686 0.11332	0.99991 0.98622 0.06991
DDOS	0.58485 0.98573 0.85307	■	0.47974 0.97834 0.81253	0.79826 0.99347 0.80162	0.75676 0.97763 0.81007	0.51072 0.98641 0.80523	0.84391 0.98467 0.80763	0.63984 0.97782 0.81488	0.61807 0.98720 0.79371
DOS	1.0 0.98192 0.87680	1.0 0.98043 0.88049	■	1.0 0.98974 0.87721	1.0 0.98541 0.86489	1.0 0.98738 0.87844	1.0 0.98276 0.88379	1.0 0.97589 0.88049	1.0 0.98425 0.86940
Injection	0.99765 0.98478 0.16363	0.33568 0.98509 0.15098	0.74627 0.98399 0.15232	■	0.81081 0.97179 0.15247	0.51110 0.98641 0.12966	0.84563 0.98467 0.15496	0.86295 0.98457 0.16645	0.65658 0.98327 0.15816
MITM	0.58514 0.98763 0.49524	0.29124 0.98788 0.44762	0.90618 0.98211 0.42857	0.71667 0.99440 0.41905	■	0.54394 0.99320 0.47619	0.84391 0.98180 0.48571	0.99749 0.98457 0.43810	0.64431 0.99213 0.41905
Password	0.99794 0.98953 0.86087	0.81842 0.98695 0.86983	0.95949 0.98305 0.87020	0.71656 0.99067 0.49627	0.59459 0.98249 0.76016	■	0.84391 0.98180 0.48571	0.99930 0.98457 0.52071	0.72774 0.99114 0.61284
Ransomware	0.99472 0.99715 0.69030	0.98927 0.98975 0.67489	0.99574 0.98399 0.09078	0.99974 0.99440 0.67591	0.97297 0.98541 0.06783	0.99991 0.99223 0.12264	■	1.0 0.99132 0.67831	0.96412 0.98819 0.09524
Scanning	0.42102 0.98002 0.91035	0.89888 0.98136 0.89017	0.77612 0.98210 0.87521	0.90636 0.98134 0.88460	0.91892 0.96887 0.87556	0.85195 0.98058 0.89066	0.84219 0.98084 0.87756	■	0.89276 0.98524 0.82944
XSS	0.99824 0.97907 0.33921	0.91483 0.98229 0.16078	1.0 0.98117 0.26518	0.99220 0.98974 0.16278	0.91892 0.97374 0.20456	0.87007 0.98641 0.27631	0.89537 0.98276 0.16875	0.99993 0.97589 0.23225	■

Table B.1: The results of using SMOTE with OCSVM model, having 50 uninteresting samples. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	1.0 0.98043 0.03949	1.0 0.97928 0.19108	1.0 0.98414 0.52918	1.0 0.9786 0.04828	1.0 0.98641 0.29843	0.99828 0.97797 0.17966	1.0 0.97589 0.18745	1.0 0.98031 0.3681
DDOS	0.99765 0.97526 0.21902	■	1.0 0.98211 0.76277	0.98007 0.98321 0.10239	0.94595 0.98152 0.36642	0.95415 0.98835 0.18779	0.85249 0.97031 0.0912	0.97360 0.97396 0.76018	0.99703 0.97244 0.01047
DOS	1.0 0.98097 0.86324	1.0 0.98043 0.86940	■	1.0 0.98507 0.86653	1.0 0.97860 0.69487	1.0 0.98641 0.85585	1.0 0.97797 0.8694	1.0 0.97396 0.86571	1.0 0.97933 0.74333
Injection	0.99794 0.98287 0.01913	0.84733 0.9795 0.02446	1.0 0.98023 0.02479	■	1.0 0.98152 0.02580	0.83272 0.98641 0.01399	0.96226 0.97414 0.02474	0.91876 0.97782 0.03241	0.93709 0.98031 0.03806
MITM	0.99824 0.97907 0.20952	0.90332 0.97856 0.08571	0.98934 0.98117 0.17143	0.94207 0.98881 0.21905	■	0.86468 0.98932 0.26667	0.87993 0.97414 0.14286	0.99406 0.97975 0.20000	0.8766 0.97343 0.04762
Password	0.99824 0.97621 0.04151	1.0 0.97763 0.00871	1.0 0.98023 0.16621	1.0 0.97854 0.00859	0.97297 0.97763 0.10808	■	1.0 0.96648 0.00847	1.0 0.97203 0.00846	1.0 0.9685 0.00833
Ransomware	0.99971 0.98097 0.02809	1.0 0.98229 0.01542	1.0 0.97928 0.01165	1.0 0.98321 0.0161	1.0 0.97568 0.02775	1.0 0.98641 0.01918	■	1.0 0.97396 0.0	1.0 0.97638 0.00343
Scanning	0.99765 0.98097 0.57846	0.93472 0.97856 0.51322	1.0 0.98023 0.50823	0.99989 0.98507 0.51232	0.97297 0.97957 0.49464	0.95402 0.98544 0.51693	0.88508 0.97414 0.50996	■	1.0 0.98031 0.50118
XSS	1.0 0.98192 0.04524	1.0 0.98043 0.0	1.0 0.98023 0.04355	1.0 0.98321 0.0	1.0 0.98152 0.0	1.0 0.98738 0.00001	1.0 0.97893 0.00006	1.0 0.97589 0.0	■

Table B.2: The results of using the OCSVM model with adding the 50 uninteresting samples as extra data. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
DDOS	1.0 0.97907 0.0	■	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
DOS	1.0 0.97907 0.0	1.0 0.98136 0.0	■	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
Injection	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	■	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
MITM	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	■	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
Password	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	■	1.0 0.97701 0.0	1.0 0.97396 0.0	1.0 0.97835 0.0
Ransomware	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	■	1.0 0.97396 0.0	1.0 0.97835 0.0
Scanning	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	■	1.0 0.97835 0.0
XSS	1.0 0.97907 0.0	1.0 0.98136 0.0	1.0 0.96987 0.0	1.0 0.98041 0.0	1.0 0.98041 0.0	1.0 0.98350 0.0	1.0 0.97701 0.0	1.0 0.97396 0.0	■

Table B.3: The results of using the OCSVM model without any data added. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		0.18423	0.06397	0.59125	0.64865	0.35004	0.80103	0.07313	0.46453
		0.98695	0.98117	0.98601	0.98249	0.99126	0.9818	0.98361	0.98425
		0.9898	0.99045	0.99051	0.99039	0.99039	0.99039	0.99027	0.99033
DDOS	0.00734		0.08955	0.67464	0.62162	0.30014	0.753	0.00049	0.53581
	0.98382		0.98493	0.98974	0.98444	0.99223	0.98563	0.98746	0.98228
	0.89153		0.91085	0.90353	0.91126	0.92743	0.89728	0.91346	0.8901
DOS	0.99794	0.49643		0.9916	0.91892	0.85653	0.97942	0.15291	0.94522
	0.96289	0.93476		0.92817	0.96984	0.9466	0.93103	0.92768	0.92618
	0.86612	0.68214		0.73388	0.88953	0.69856	0.62998	0.7614	0.66201
Injection	0.08808	0.15356	0.11514		0.72973	0.54005	0.79931	0.12147	0.66951
	0.9686	0.96831	0.96893		0.96984	0.97573	0.96935	0.9595	0.96063
	0.12394	0.08425	0.18378		0.16943	0.09034	0.12024	0.11631	0.09211
MITM	0.00734	0.04982	0.60768	0.66025		0.46584	0.77358	0.03562	0.61673
	0.99334	0.98695	0.98588	0.98787		0.99029	0.98659	0.9865	0.9813
	0.47619	0.45714	0.46667	0.45714		0.4381	0.45714	0.46667	0.4381
Password	0.95684	0.75648	1.0	0.99284	0.89189		0.95883	0.98771	0.99612
	0.98002	0.97856	0.9774	0.98694	0.98152		0.98372	0.97975	0.98031
	0.26634	0.1172	0.26996	0.27682	0.28541		0.28843	0.27196	0.24122
Ransomware	0.0	0.1015	0.04051	0.07914	0.27027	0.05419		0.00021	0.21377
	0.9981	0.99627	0.99435	0.9972	0.99611	0.99806		0.99711	0.99705
	0.92155	0.91881	0.92121	0.90373	0.93114	0.92326		0.93354	0.91847
Scanning	0.99971	0.882	0.82303	0.99612	0.89189	0.97301	0.9554		0.95324
	0.97241	0.93942	0.93597	0.92164	0.97082	0.96019	0.97031		0.92126
	0.81699	0.81636	0.81636	0.81636	0.81957	0.81647	0.81653		0.81636
XSS	0.24868	0.15602	0.10448	0.87706	0.72973	0.38023	0.76158	0.00084	
	0.93054	0.93663	0.93032	0.91698	0.95817	0.94369	0.9454	0.92575	
	0.31655	0.27462	0.3356	0.30581	0.43519	0.31832	0.41477	0.33594	

Table B.4: The results of using SMOTE with IForest model, having 50 uninteresting samples. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		0.99889 0.98043 0.09097	0.99147 0.98305 0.66913	0.99996 0.98414 0.0	0.83784 0.98249 0.64957	0.99994 0.99126 0.79525	1.0 0.98372 0.75887	0.99986 0.9865 0.79502	1.0 0.97343 0.0
DDOS	0.42337 0.98192 0.0	-	1.0 0.98305 0.00018	0.99981 0.98881 0.00132	0.94595 0.98346 0.0065	1.0 0.99223 0.00005	1.0 0.98467 0.00026	1.0 0.98168 0.00037	0.99994 0.98228 0.00112
DOS	0.99853 0.98097 0.00084	0.99974 0.98229 0.00042		0.99989 0.98321 0.00042	0.86486 0.98444 0.01677	0.99997 0.99223 0.00461	1.0 0.98563 0.01048	0.99986 0.98457 0.02013	1.0 0.97638 0.00042
Injection	1.0 0.98287 0.00008	0.99948 0.98229 0.00 38	1.0 0.98399 0.00148		0.91892 0.98833 0.01378	1.0 0.99223 0.00051	1.0 0.98563 0.00339	0.99993 0.98843 0.02653	0.99997 0.98031 7e-05
MITM	1.0 0.98668 0.0	1.0 0.98229 0.0	0.99787 0.98682 0.0	0.99996 0.98787 0.0		1.0 0.9932 0.0	1.0 0.99042 0.10909	0.99986 0.99036 0.10909	1.0 0.98031 0.0
Password	0.98238 0.98097 0.0	0.99995 0.98229 0.00001	1.0 0.98399 0.00006	0.99992 0.98787 0.00001	0.97297 0.98346 0.00034		1.0 0.98372 0.00004	1.0 0.98457 0.00004	0.99997 0.97835 1e-05
Ransomware	1.0 0.98668 0.0	0.99998 0.98602 0.0	1.0 0.98305 0.0	0.99992 0.98601 0.0	0.94595 0.98735 0.00035	1.0 0.9932 0.0		1.0 0.98746 0.0	1.0 0.98031 0.0
Scanning	1.0 0.97716 0.00968	0.94363 0.98043 0.00793	0.99147 0.98117 0.01477	0.99928 0.98321 0.0	0.83784 0.98152 0.31901	0.97899 0.99029 0.0	1.0 0.9818 0.06729		0.99994 0.97343 0.0
XSS	0.16882 0.98097 0.08763	0.99263 0.98136 0.02964	1.0 0.97269 0.00041	0.99955 0.98414 0.02252	1.0 0.97665 0.00005	0.99984 0.99029 0.05896	1.0 0.98372 0.05708	0.9993 0.97975 0.01501	

Table B.5: The results of using the IForest model with adding the 50 uninteresting samples as extra data. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.00012	0.94595 0.97471 0.00116	0.99984 0.98738 0.04424	1.0 0.97989 0.0	1.0 0.97878 0.06733	0.99994 0.9685 0.0
DDOS	1.0 0.9705 0.0		1.0 0.96893 0.0	1.0 0.97481 0.0	0.94595 0.97471 0.0	0.99984 0.98738 0.0	1.0 0.97989 0.0	1.0 0.97878 0	0.99994 0.9685 0.0
DOS	1.0 0.9705 0.0	1.0 0.97577 0.0		1.0 0.97481 0.00287	0.94595 0.97471 0.0	0.99984 0.98738 0.0	1.0 0.97989 0.0	1.0 0.97878 0	0.99994 0.9685 0.0
Injection	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0		0.94595 0.97471 0.0	0.99984 0.98738 0.0	1.0 0.97989 0.0	1.0 0.97878 0.06733	0.99994 0.9685 0.0
MITM	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.0	0.97471	0.99984 0.98738 0.04424	1.0 0.97989 0.0	1.0 0.97878 0	0.99994 0.9685 0.0
Password	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.0	0.94595 0.97471 0.0		1.0 0.97989 0.0	1.0 0.97878 0	0.99994 0.9685 0.0
Ransomware	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.0	0.94595 0.97471 0.0	0.99984 0.98738 0.0		1.0 0.97878 0	0.99994 0.9685 0.0
Scanning	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.0	0.94595 0.97471 0.0	0.99984 0.98738 0.0	1.0 0.97989 0.0		0.99994 0.9685 0.0
XSS	1.0 0.9705 0.0	1.0 0.97577 0.0	1.0 0.96893 0.0	1.0 0.97481 0.0	0.94595 0.97471 0.0	0.99984 0.98738 0.0	1.0 0.97989 0.0	1.0 0.97878 0.06733	

Table B.6: The results of using the OCSVM model without any data added. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		0.99956 0.96179 0.40338	1.0 0.97175 0.40355	0.99872 0.95709 0.40226	0.97297 0.97763 0.40449	0.98823 0.98155 0.40367	0.99314 0.97414 0.40379	0.99993 0.97107 0.40408	0.97474 0.95768 0.40291
DDOS	0.86436 0.92959 0.11065		0.71855 0.95386 0.68262	0.9797 0.96175 0.20429	0.78378 0.98249 0.84911	0.77588 0.96602 0.65059	0.84391 0.98084 0.84388	0.88467 0.94793 0.08257	0.95216 0.94291 0.06696
DOS	0.99765 0.97241 0.89158	0.99997 0.95433 0.89322		0.84843 0.95616 0.90062	0.75676 0.98444 0.9117	0.97292 0.95728 0.89405	0.99828 0.95881 0.89405	0.98652 0.93925 0.88912	0.99974 0.95177 0.88994
Injection	0.99765 0.96765 0.06529	0.23558 0.94129 0.0763	0.58209 0.95009 0.06034		0.78378 0.98346 0.16978	0.69401 0.96505 0.04465	0.8422 0.98946 0.46198	0.82006 0.94407 0.05441	0.90555 0.94783 0.05943
MITM	0.99706 0.9334 0.11429	0.52133 0.94129 0.06667	0.94883 0.96704 0.4381	0.92678 0.95709 0.09524		0.89995 0.96019 0.12381	0.8422 0.97989 0.79048	0.98806 0.93539 0.11429	0.77655 0.94882 0.11429
Password	0.99736 0.97241 0.85617	0.43966 0.94688 0.47941	0.97655 0.96798 0.87994	0.8461 0.95522 0.40594	0.81081 0.9679 0.9023		0.84391 0.98946 0.97024	0.99923 0.96528 0.64519	0.84626 0.95177 0.51831
Ransomware	0.99941 0.98002 0.08976	0.99992 0.98136 0.074	1.0 0.98211 0.08051	0.99959 0.9916 0.07537	1.0 0.98249 0.07537	0.99997 0.99223 0.05995		1.0 0.98457 0.07948	0.99989 0.98524 0.07571
Scanning	0.9956 0.94481 0.93059	0.43721 0.94408 0.83668	0.47974 0.94539 0.86886	0.9942 0.95896 0.92268	0.83784 0.97568 0.9556	0.7859 0.95922 0.93371	0.83533 0.98467 0.97172		0.90555 0.94783 0.05943
XSS	0.99648 0.94577 0.06568	0.9347 0.94967 0.04995	0.58209 0.95104 0.05733	0.96859 0.96455 0.07058	0.78378 0.98152 0.29647	0.55194 0.95825 0.27525	0.84391 0.98946 0.36422	0.71766 0.94503 0.05889	

Table B.7: The results of using SMOTE with VAE model, having 50 uninteresting samples. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	0.99977 0.96552 0.40015	1.0 0.97363 0.40220	0.99985 0.97108 0.39347	0.94595 0.98054 0.39793	0.99409 0.96117 0.38316	1.0 0.97031 0.40056	1.0 0.97396 0.40191	0.99986 0.93732 0.02904
DDOS	0.99765 0.97812 0.76345	■	1.0 0.97175 0.06592	0.99906 0.95989 0.02397	0.86486 0.97665 0.77546	0.99988 0.9699 0.01407	1.0 0.95594 0.00841	1.0 0.95082 0.00164	0.96443 0.95276 0.04492
DOS	0.99765 0.97812 0.8961	1.0 0.96645 0.67474	■	1.0 0.97575 0.87228	0.91892 0.98054 0.89774	1.0 0.98058 0.86858	0.99485 0.97414 0.80657	1.0 0.96239 0.79014	0.99994 0.97539 0.89076
Injection	0.99765 0.97431 0.10593	1.0 0.96925 0.0	0.99787 0.97269 0.01413	■	0.78378 0.98152 0.17478	0.99714 0.96019 0.00091	1.0 0.93966 0.00026	1.0 0.96336 0.0019	0.99883 0.95472 0.00147
MITM	0.99765 0.98097 0.21905	0.99997 0.97018 0.0	0.99787 0.97081 0.02857	1.0 0.97295 0.0	■	0.99994 0.98447 0.09524	1.0 0.96839 0.0	1.0 0.97011 0.00952	0.99997 0.97343 0.0
Password	0.99736 0.98382 0.62739	0.99998 0.96645 0.00463	1.0 0.97834 0.24065	1.0 0.97948 0.02993	1.0 0.97082 0.0288	■	1.0 0.97222 0.02917	0.99958 0.97396 0.1309	0.99989 0.97736 0.00096
Ransomware	0.99971 0.98953 0.09695	0.99997 0.99068 0.07023	1.0 0.98588 0.07811	1.0 0.99347 0.01816	0.91892 0.99027 0.37205	1.0 0.9932 0.61425	■	1.0 0.99036 0.15245	0.99994 0.99409 0.07571
Scanning	0.99765 0.96955 0.68071	0.95237 0.95061 0.38923	0.99787 0.95669 0.36464	0.99989 0.95896 0.16817	0.97297 0.97568 0.58736	0.82388 0.95631 0.37736	0.98285 0.95594 0.43123	■	0.95875 0.95374 0.37001
XSS	0.99765 0.97146 0.27746	0.99618 0.95247 0.02841	1.0 0.96987 0.04382	0.9968 0.96735 0.04539	0.94595 0.97179 0.06189	0.97902 0.96214 0.03561	0.99485 0.96456 0.04229	0.99986 0.93732 0.02904	■

Table B.8: The results of using the VAE model with adding the 50 uninteresting samples as extra data. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
DDOS	1.0 0.99905 0.0	■	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
DOS	1.0 0.99905 0.0	1.0 0.9972 0.0	■	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
Injection	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	■	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
MITM	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	■	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
Password	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	■	1.0 0.99904 0.0	1.0 0.99036 0.0	1.0 1.0 0.0
Ransomware	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	■	1.0 0.99036 0.0	1.0 1.0 0.0
Scanning	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	■	1.0 1.0 0.0
XSS	1.0 0.99905 0.0	1.0 0.9972 0.0	0.99787 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 1.0 0.0	1.0 0.99904 0.0	1.0 0.99036 0.0	■

Table B.9: The results of using the VAE model without any data added. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		0.99979 0.95620 0.40045	1.0 0.97834 0.40221	0.99872 0.95989 0.40186	0.91892 0.97374 0.40168	0.98817 0.97864 0.40203	0.99142 0.97414 0.40221	1.0 0.96914 0.40215	0.95333 0.95571 0.40197
DDOS	0.99736 0.96289 0.79334		1.0 0.95104 0.76964	0.95793 0.96175 0.77761	0.78378 0.97568 0.80299	0.79455 0.95922 0.75676	0.94683 0.93199 0.10772	0.87727 0.94503 0.76599	0.93523 0.94980 0.77379
DOS	0.99765 0.97050 0.89392	1.0 0.9534 0.89224		0.98960 0.95243 0.89308	0.78378 0.97957 0.90231	0.99988 0.96408 0.89099	0.84391 0.98659 0.90566	1.0 0.95757 0.89476	0.99852 0.95079 0.89140
Injection	0.99736 0.96860 0.63338	0.26042 0.94781 0.41038	0.97655 0.95857 0.44364		0.72973 0.96693 0.52383	0.51421 0.96311 0.39977	0.84391 0.98946 0.69899	0.88384 0.94793 0.25053	0.90971 0.95374 0.40128
MITM	0.99853 0.94481 0.23636	0.48646 0.94501 0.09091	0.95736 0.95669 0.30909	0.60206 0.96175 0.36364		0.64454 0.96505 0.76364	0.8319 0.98946 0.98182	0.99923 0.95082 0.34545	0.73365 0.95276 0.34545
Password	0.99736 0.95433 0.92682	0.91744 0.94967 0.89985	0.96162 0.97175 0.90712	0.65622 0.95802 0.52357	0.81081 0.95428 0.74031		0.91252 0.94253 0.67048	0.99923 0.96914 0.8543	0.78172 0.95276 0.52507
Ransomware	0.99912 0.98287 0.14221	1.0 0.98322 0.09969	1.0 0.98305 0.19798	0.99996 0.98974 0.20042	0.97297 0.98152 0.83025	0.99988 0.99029 0.14918		1.0 0.97686 0.06274	0.99997 0.96949 0.16661
Scanning	0.99765 0.96575 0.92523	0.93482 0.95061 0.79261	0.30277 0.95198 0.92274	0.99401 0.96175 0.84866	0.89189 0.97374 0.89656	0.78795 0.95825 0.88510	0.83533 0.98563 0.97119		0.94394 0.95177 0.88832
XSS	0.99765 0.96860 0.21180	0.93470 0.95806 0.05037	0.56290 0.95104 0.05648	0.97691 0.95616 0.05893	0.89189 0.97179 0.1562	0.52299 0.95437 0.31157	1.0 0.94444 0.04550	0.64816 0.93635 0.11031	

Table B.10: The results of using SMOTE with AE model, having 50 uninteresting samples. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor	■	0.99988 0.95713 0.40045	1.0 0.97269 0.40062	0.99947 0.96828 0.40045	0.89189 0.9786 0.40315	0.9882 0.98155 0.40045	1.0 0.97222 0.40045	1.0 0.96625 0.40045	0.98944 0.95374 0.40045
DDOS	0.99736 0.96384 0.8201	■	1.0 0.9661 0.78159	0.99725 0.96269 0.1429	0.78378 0.97471 0.83251	0.97295 0.96699 0.58529	0.92453 0.9454 0.06606	0.97045 0.95082 0.13227	0.95073 0.95079 0.57478
DOS	0.99765 0.96384 0.8956	1.0 0.96925 0.88721	■	1.0 0.97201 0.89224	0.89189 0.97568 0.89686	1.0 0.98641 0.8914	1.0 0.9751 0.89266	0.97129 0.94696 0.88134	0.94936 0.95669 0.88679
Injection	0.99765 0.96004 0.43255	0.99989 0.95433 0.11476	0.95522 0.94727 0.61243	■	0.89189 0.9572 0.39619	0.98431 0.96408 0.14125	1.0 0.97031 0.22619	0.94419 0.94118 0.00958	0.79479 0.9498 0.27504
MITM	0.99736 0.98002 0.30909	0.92964 0.97928 0.94545	0.99147 0.96704 0.09091	0.99996 0.97015 0.10909	■	0.99972 0.98932 0.10909	1.0 0.9569 0.0	0.99944 0.96046 0.32727	0.99983 0.9439 0.07273
Password	0.99736 0.97716 0.90408	0.96741 0.94688 0.70583	0.98294 0.96422 0.62425	0.74726 0.96455 0.49294	0.7027 0.96012 0.90988	■	0.84391 0.98276 0.95826	0.99958 0.96818 0.47091	0.65179 0.95177 0.67301
Ransomware	1.0 0.99049 0.07529	0.99997 0.98975 0.04705	1.0 0.98305 0.07389	1.0 0.99534 0.06727	1.0 0.98346 0.06588	1.0 0.99223 0.07389	■	1.0 0.98361 0.07494	0.99997 0.99016 0.0962
Scanning	0.99765 0.96384 0.75964	0.93482 0.95527 0.48401	1.0 0.94915 0.71116	0.99217 0.96362 0.8085	0.78378 0.97179 0.87441	0.80588 0.95825 0.87928	0.9211 0.93966 0.76149	■	0.94168 0.95768 0.83746
XSS	0.99765 0.9705 0.16006	0.9347 0.96086 0.05033	0.98081 0.95009 0.04557	0.99635 0.96269 0.04896	0.78378 0.97471 0.34465	0.97295 0.95922 0.04557	1.0 0.95211 0.04905	0.89578 0.94407 0.04897	■

Table B.11: The results of using the AE model with adding the 50 uninteresting samples as extra data. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting.

	Backdoor	DDOS	DOS	Injection	MITM	Password	Ransomware	Scanning	XSS
Backdoor		1.0 0.99814 0.0	0.99787 0.99812 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	0.99994 1.0 0.0	1.0 0.99808 0.0	1.0 0.99711 0.0	1.0 1.0 0.0
DDOS	1.0 0.99905 0.0		1.0 0.99718 0.0	1.0 1.0 0.0	1.0 0.99708 0.0	1.0 0.99903 0.0	1.0 0.99904 0.0	1.0 0.99229 0.0	1.0 1.0 0.0
DOS	1.0 0.99905 0.00335	1.0 0.99814 0.0021		1.0 1.0 0.00545	1.0 0.99805 0.00294	0.99994 1.0 0.00545	1.0 0.99904 0.0	1.0 0.99421 0.0	1.0 1.0 0.00545
Injection	1.0 0.99905 0.0	1.0 0.99814 0.0	0.99787 0.99435 0.0		1.0 0.99805 0.0	0.99994 1.0 0.0	1.0 0.99904 0.0	1.0 0.99614 0.0	1.0 1.0 0.0
MITM	1.0 0.9981 0.0	1.0 0.99814 0.0	1.0 0.99623 0.0	1.0 1.0 0.0		0.99994 1.0 0.0	1.0 0.99808 0.0	1.0 0.98939 0.0	1.0 1.0 0.0
Password	1.0 0.9981 0.00003	1.0 0.9972 0.0001	0.99787 0.99812 0.00002	1.0 1.0 0.00009	1.0 0.99708 0.00003		1.0 0.99904 0.00002	1.0 0.99711 0.00001	1.0 1.0 0.00009
Ransomware	1.0 0.99905 0.0	1.0 0.99627 0.0	0.99787 0.99812 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	1.0 0.99903 0.0		1.0 0.99614 0.0	1.0 1.0 0.0
Scanning	1.0 0.9981 0.00392	1.0 0.99814 0.00049	1.0 0.99435 0.0	1.0 1.0 0.02519	1.0 0.99805 0.00049	1.0 0.99903 0.00049	1.0 0.99904 0.00049		1.0 1.0 0.05497
XSS	1.0 0.99905 0.0	1.0 0.9972 0.0	1.0 0.99718 0.0	1.0 1.0 0.0	1.0 0.99805 0.0	0.99994 1.0 0.0	1.0 0.99904 0.0	1.0 0.99711 0.0	

Table B.12: The results of using the AE model without any data added. The first value denoted is the accuracy for the anomaly class, the second for the regular data and the third for the uninteresting anomaly being labelled as uninteresting

Model	Which method	Accuracy Anomalies	Accuracy regular data	Accuracy uninteresting anomolies
IForest	No extra data	**0.7566**	**0.9354**	0
IForest	Few extra	0.7536	0.9214	0.3265
IForest	Resampling SMOTE	0.5574	0.6587	**0.8426**
AE	No extra data	**0.6458**	**0.7896**	0.0044
AE	Few extra	0.5544	0.5789	0.1545
AE	Resampling SMOTE	0.2447	0.6584	**0.7369**
OCSVM	No extra data	**0.7569**	**0.9368**	0
OCSVM	Few extra	0.6935	0.9144	0.1210
OCSVM	Resampling SMOTE	0.4578	0.8756	**0.7681**
VAE	No extra data	**0.8265**	**0.8900**	0.0745
VAE	Few extra	0.8458	0.9271	0.5413
VAE	Resampling SMOTE	0.6759	0.8464	**0.7459**

Table B.13: The performance of the different models using our framework on the healthcare dataset, where inconclusive is the uninteresting anomaly and abnormal interesting. As can be seen from the results, all of the different models perform relatively similarly on the baseline of no extra data. Highlighted are per model which method gave the best accuracy per class.

C Results experiment 2

In this chapter the additional plots for experiment 2 are shown.

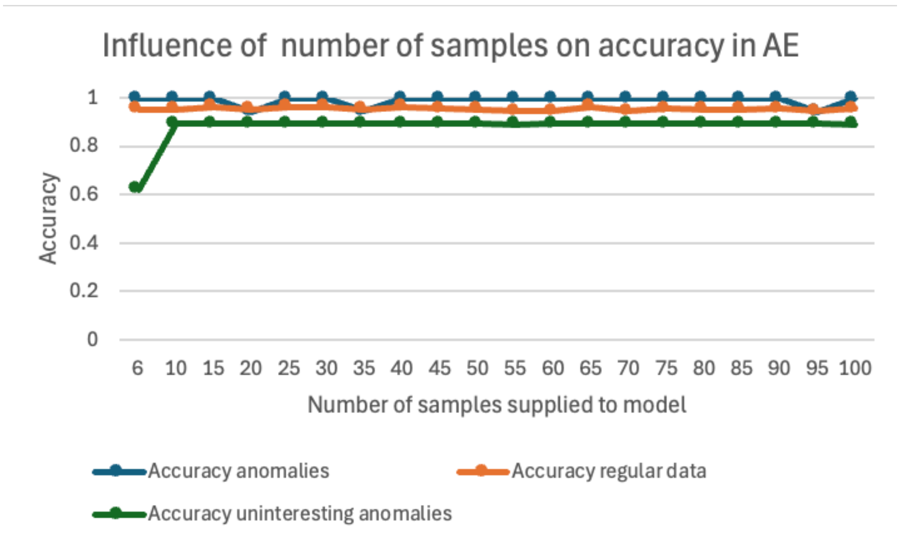


Figure C.1: The number of samples needed to create reliable results for Autoencoders. Here we can see that the model is very consistent, and for this combination of model and interesting and uninteresting anomaly produces very reliable results with not much variance.

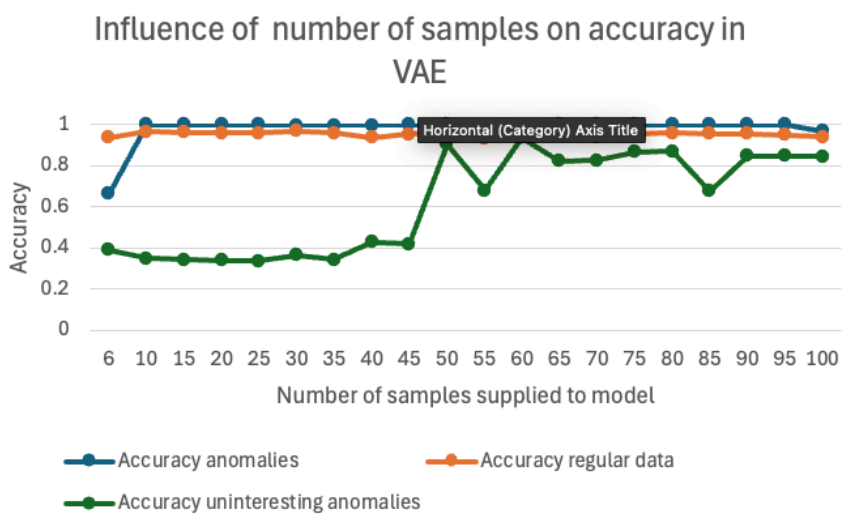


Figure C.2: The number of samples needed to create reliable results for Variational Autoencoders. Here we can see that the model needs more time to learn data, only producing good results when we near the 100 samples gathered.

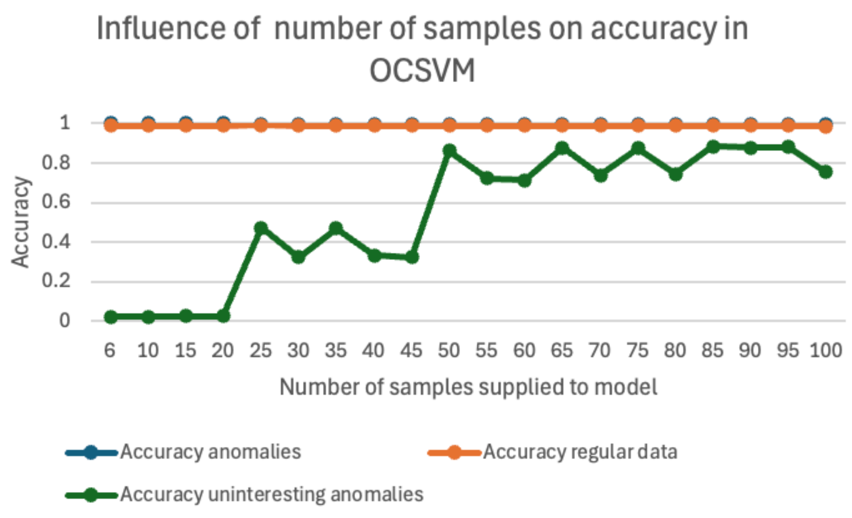


Figure C.3: The number of samples needed to create reliable results for OCSVM. Here we can see that the model needs more time to learn data, only producing good results when we near the 50 samples gathered.