Towards Interpretable Anomaly Analytics for Secure Testing Environments

Spee, H.M.

Award date:
2020

Department of Mathematics and Computer Science
Data Mining Research Group

Towards Interpretable Anomaly Analytics for Secure Testing Environments

Master Thesis

Hugo Spee 0812503

Supervisors:
prof. dr. Mykola Pechenizkiy (TU/e)
MSc. Thonie van Lieburg (Carapax IT)
dr. Jerry den Hartog (TU/e)

version 1.1

Eindhoven, September 2020

Abstract

Many students at the Technische Universiteit Eindhoven and the Avans Hogeschool use the secure exam environment STEP to make their exams on their own computers within a secured custom Linux distribution. It is important that there is some automated way to detect fraud and faults in logs generated by STEP, such that the integrity of the examination process remains intact. Potential machine learning solutions are explored for anomaly detection for secure exam environments. A deep learning bidirectional LSTM for predicting the probability distribution of the vocabulary trained on non-anomalous logs is the most promising solution. It is attempted to add methods from the interpretability domain, however an existing model can not be found that works with the output of the bidirectional LSTM. Instead a visualization in Excel is created to visualize the anomaly scores.

Preface

Three years ago, just before starting my master's thesis, I developed agoraphobia–(extreme) stress in open spaces–which caused me to stop being able to go to the university. Due to this I had difficulties working on my master's thesis. My original subject direction for my master thesis was a continuation of my seminar project on reinforcement learning during which I was supervised by dr. ir. Joaquin Vanschoren. We decided that this project was not working for me and I switched my master's thesis' subject to "Automatic Jupyter Notebook Generation for OpenML". A project where I attempted to automatically generate Jupyter Notebook documents from OpenML datasets, OpenML.org being an online platform for hosting datasets, models and experiments founded by Joaquin. This project did not work out in the end because I decided that the quality of the work was not high enough by my standards. I then searched for a new subject for my master's thesis in a better environment where I could get more help and discussion about my work. It was then that Carapax IT–an IT company that focuses on autistic employees–offered me the opportunity to work on a problem from Secure Testing B.V., one of their clients. I started the project under the supervision of prof. dr. Mykola Pechenizkiy and MSc. Thonie van Lieburg as my company supervisor. Unfortunately a pandemic hit the world as I was finishing off my master's thesis, causing further unforeseen delays. But I am happy to say that you now finally have my completed thesis in front of you.

I would like to thank the following people who helped me throughout my journey. First and foremost I would like to thank Mykola Pechenizkiy and Thonie van Lieburg as my supervisors, they helped me finally get through the difficulties I had and finish my master's thesis. I would also like to thank the members of the exam committee. From Carapax IT I would like to thank Monique for keeping me on track, Marcel from the STEP team for showing me the ropes of the system and Bram for his help and feedback. Very special thanks to René from the STEP team for thinking with me, helping me throughout the project with STEP related questions and his feedback. I would also like to thank Huub from Secure Testing B.V. for providing the business problem. From the TU/e I would also like to thank Joaquin Vanschoren for his help as my first supervisor, dr. Natasha Stash and Ms. Katie MacLeod for their help in their capacity as academic advisor for master students. I would like to thank my father and mother, Karel and Susan for proofreading my master's thesis and supporting me throughout, as well as my sister Jitske. Finally, I would like to thank my activity supervisor Mary who has helped me decrease my agoraphobia and helped me to go out again and work on my thesis at Carapax IT.

This master's thesis finally marks the end of my journey to finally graduate. It took longer than expected but we have finally reached the destination. I hope you will enjoy reading this master's thesis, and appreciate the journey it took to finally get to this point. As Brandon Sanderson states in his book *The Way of Kings* - *"Journey before Destination."*, but I am still happy to have finally reached the destination.

Hugo Spee

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Anomalous data can indicate interesting non-standard behaviour within a computer system, analysis of this data can provide clues for failures of the system and non-proper usage amongst other uses. This thesis focuses on developing a machine learning framework for detecting anomalies within a custom Linux distribution on a USB stick that limits the capabilities of the system to which the USB stick is connected. This Linux distribution is used during exams to allow students to use their own laptop within a walled garden environment and is henceforth referred to as STEP. It is therefore useful to be able to detect failures within this Linux distribution and possible attempts by students to bypass the walled garden as well as explain why these failures happened and how the walled garden was bypassed.

1.1 Background and Business Case

The TU/e based start-up company Secure Testing B.V., in partnership with the IT company Carapax IT, has developed a secure Linux based operating system that is loaded onto a USB stick. These USB sticks are given to students to be used during exams, they will insert the USB sticks into their laptops which gives them access to their laptop during the exam within a secured environment. Because Secure Testing B.V. is selling their system to different educational institutions they would like to include some method of detecting patterns or instances in their data which do not conform to expected behaviour, i.e. anomalies. Anomalies can help the educational institutions identify where certain faults in the system originate, as well as identifying cases of fraud within the system. Faults of the STEP system are defined as failures in software and/or hardware which occur when a computer system is running the STEP System. The cause of a fault is logged in Linux and/or STEP logs, but might not be easy to find for a layperson. Fraud in the STEP system is defined as a malicious attack on the STEP system by a bad faith actor. When a malicious attack on the STEP System succeeds, this will allow the bad faith actor to cheat on an examination. Having an anomaly detection system will allow Secure Testing B.V. to more efficiently provide support by in-house experts and a way to more easily investigate suspicious behaviour. Just having an anomaly detection system, however, does not mean that it will be clear why something was flagged as an anomaly. Because the system will be used by lay people and software engineers from Carapax IT, it is useful to offer more insight into why something was flagged as an anomaly. Therefore it would be useful to add a interpretable anomaly alert processing framework which can explain the decision of an algorithm in a way that a human can understand it. Moreover, in Europe the General Data Protection Regulation (GDPR) provides its citizens with the "right to explanation" (Goodman and Flaxman, 2017), the right to

be provided with an explanation on why an important decision regarding yourself was taken with help of an algorithm. So, if a student is penalized for having committed fraud, based on a decision reached using the system, and this student asks for an explanation, it is ideal if the system is able to provide the end users with an explanation for why a certain log was flagged. It is important to stress that in the end a human will make decision on what to do when the system flags an anomaly. If a student takes the educational institutions using the secure testing environment to court over a decision made by the model, it is important that the decision can be explained by the model such that it can hold up in court. Preferably, therefore, any fraud detection system should be GDPR compliant. In recent times, with the ongoing 2019–20 coronavirus pandemic massive quarantines are deployed by governments around the world to prevent the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) from spreading. Due to these quarantines many examinations will be held online and dealing with fraud in these cases is expected to become an important problem in the foreseeable future. The need for research into interpretable secure testing environments therefore becomes more important than ever. This work is divided into two distinct parts, an anomaly detection framework and an interpretable anomaly alert processing framework.

1.2 Problem Context

Secure Testing B.V. would like to improve and further integrate their methods of detecting faults and fraud attempts. As the STEP system is used for administering exams by educational institutions, Secure Testing B.V. would like to be able to further assure their clients that the STEP system can inform administrative users of fraud attempts and faults caused by new computer configurations. Despite the wish of Secure Testing B.V. to specifically detect fraud attempts, it was decided to limit the scope of this problem to finding any anomaly within a set of Linux logs. This is due to the secretive nature of the fraud detection domain which limits access to information about the state of the art of detection systems specifically designed to detect fraud. The state of the art of more general anomaly detection systems is less secretive and therefore it was decided to limit the scope to the more general task of finding anomalies rather than finding fraud attempts in particular. Our goal is creating a machine learning model which can:

1. Find anomalous sessions[1] i.e. sessions that contain faults or fraud attempts.

2. Find and visualize anomalous log-lines within an anomalous session.

Furthermore, since every educational institution has its own version of the STEP system the model should be able to provide the end users who are not domain experts with enough information such that they can self-diagnose problems or know when to contact Secure Testing B.V. if there are issues in their system requiring expert support.

1.2.1 STEP system

The STEP system (Secure Testing Environment Protocol) is a Linux based operating system that is loaded onto a USB stick. When inserted into a computer the computer boots into a secure environment with limited access to the internet and tools like calculators. Educational institutions are able to select which internet domains they provide access to and which tools they wish to include on the OS for each individual USB stick. The goals of this project are

[1] A session is the set of all logs that have been generated by the STEP system for a single USB stick entered into a student machine from USB startup to shutdown. See 3.3.1 for the detailed definition.

twofold, first is to apply anomaly detection on the collected logs to detect faults and cases of fraud. Secondly the goal of this project is to apply interpretable anomaly alert processing on the results of the anomaly detection to make the results easier to understand for a human and to allow faster debugging of the faults and potential fraud cases found in the system.

1.2.2 Stakeholders

Three groups of stakeholders for the anomaly detection system are identified, as shown in Fig. 1.1:

- STEP software engineer and testers at Carapax IT.
- Exam Committee members at educational institutions.
- Technical staff at educational institutions.

STEP technical expert and tester at Carapax IT

The STEP Technical Expert of Carapax IT is the programmer behind the STEP system. He is an expert in Linux systems and can manually find issues in Linux logs. There is also a software tester who has less knowledge of Linux but can also find issues in logs, although not at the same level as the technical expert. They are collectively referred to as the experts of Carapax IT. Carapax IT often receives logs from Secure Testing B.V containing hardware issues. These logs come from laptops which have experienced faults while using the STEP system and do not contain personally identifiable information. Currently the experts have to go through each of these logs manually to check where the fault occurs, which is very time consuming. Faults that are found manually are annotated and fixed when possible. The experts would like a faster way to find faults in these logs, so they do not have to spend so much time on manually going over each log.

Exam Committee members at educational institutions

The educational institutions are currently using the STEP system and want to assure that if they award someone with a diploma, this person verifiably meets the standards required for that diploma. The educational institutions do not want to devalue the credibility of their diplomas if suspicions are substantiated that it is possible to cheat your way through certain exams. Hence they want a way to identify fraud attempts within sessions generated during their exams. If the exam commitee suspects someone of committing fraud during an exam, for example if an exam invigilator suspects that fraud might have occurred, based on their surveillance, then Secure Testing B.V. expects that the exam committee could request Secure Testing B.V. to check specific log files associated with the exam for fraud, if the educational institutions have issues figuring out if there is fraud in the logs themselves. These log files will then be sent to Carapax IT by Secure Testing B.V. and the technical expert and tester of Carapax IT will then be asked to find evidence of fraud within the logs. Having a system that would mark potentially fraudulent lines would be useful for the technical expert and tester of Carapax IT in providing additional evidence to substantiate the fraud claims to the exam committee, as accusing someone of fraud is a serious matter. The technical expert and tester of Carapax IT will only indicate to the exam committee which log-lines are anomalous, the decision whether or not the found log-lines indicate indictable fraud is left to the exam commitee.

Technical staff at educational institutions

If something goes wrong with a laptop running STEP during an exam the technical staff of the educational institution will be informed. The technical staff will send clean new logs of the laptops that had problems running STEP to Secure Testing B.V. which will ask the expert and tester of Carapax IT to find the faults within the logs and fix them. Because the time of the technical expert of Carapax IT is valuable and there might be a lot of logs of faulty laptops, it is useful for there to be a method for the technical expert of Carapax IT to find faults faster.

1.2.3 Considerations for the Evolution of the STEP Project in the Future

It is important to note that STEP is still a system in development; as such the business requirements and stakeholders might change over time. Right now it is assumed that logs are reported by skilled staff at the TU/e but in the future, as STEP is sold to more educational institutions, the use case presented in Fig. 1.1 might change. Note that, while STEP is used by Avans Hogeschool too, in this report we will focus primarily on the Tu/e only, as the TU/e is where we get most of our logs from. It is reasonable to expect that finding anomalous user sessions within a group of sessions (inter-log anomaly detection) in addition to anomalous log-lines (intra-log anomaly detection) will become an important business goal in the future. It is also reasonable to expect that the technical expert of Carapax IT will not be able to go through logs for the lifetime of the STEP system, therefore any solution should also be usable for people with less technical knowledge, such as employees of educational institutions.

1.2.4 Anticipated Utility of Research Results

The anticipated utility of the research results is as follows:

- A machine learning model and machine learning pipeline is provided that finds anomalies within Linux logs (intra-log anomaly detection) which can be implemented into STEP.
- A machine learning model and machine learning pipeline that finds Linux logs that are anomalous (inter-log anomaly detection) is provided.
- Users can be notified of suspicious logs.
- A visualization of the model alerts is provided which is suitable for a lay person and aids the experts at Carapax IT in finding faults and anomalies within the logs.

1.2.5 Business Requirements

As this is an exploratory work into the feasibility of anomaly/fraud detection for the STEP system there are no strict requirements for the approach from the company side. The best case scenario for the client would be to have an approach that can be integrated into STEP. This approach should find logs with faults/fraud automatically with high true positives rates and low false positives rates and produce an easy to use overview of the faults/fraud. Since no requirements were given for performance of the model the goal of this study will be to find the best possible approach within the time limit given. The success of the project in terms of business goal will ultimately be decided by the customer outside the scope of this work. Furthermore, regarding other business requirements, any machine learning algorithm is allowed to be picked and no requirements regarding processing time were given. The analysis may take place after an

exam is over and all data has been gathered, therefore at this moment it is not a goal to perform data analysis during an exam in near real-time.

1.3 Formulation, Scope and Approach

1.3.1 Approach

An overview of the data analytics framework is given in Fig. 1.1.

Anomaly Analytics for Exam Fraud Detection

STEP logs are received, they are preprocessed and then sent to the anomaly detector which generates anomaly scores. Manually labeled logs by the technical expert of Carapax IT can be used to evaluate intra-log detection performance. All logs are labeled for being normal or containing faults/hack attempts, which can be used to evaluate inter-log detection performance. Carapax IT employs data engineers who can maintain the machine learning pipeline.

Interpretable Anomaly Alert Processing

The alert processing unit receives the anomaly scores from the anomaly detector and uses Explainable AI or a visualization to add a measure of explainability such that the technical expert and tester of Carapax IT can understand the results faster which helps them save precious time. The solution is evaluated by the technical expert and the tester of Carapax IT for its usability in simplifying their work.

1.3.2 Research Formulation

The main research question is:

> *How should an interpretable anomaly detection framework be created that detects system faults and fraud cases and then provides users with valuable information about those system faults and fraud cases.*

In order to answer this question this research question is split into two sub-questions:

> *How should an anomaly detection framework be created that can rank the content of Linux logs and user data on their abnormality in comparison with known normal Linux logs and user data. As well as detect and generate an anomaly alert for those Linux logs and user data which are likely to contain system faults or fraud cases based on their ranking?*

and

> *How should an interpretable anomaly alert processing framework be created that provides an overview of the detected anomalies, as well as providing a human understandable explanation of the reasons that these anomalies were detected by the anomaly detection framework.*

These two research questions encompass the two core domains of this study: anomaly detection and interpretable anomaly alert processing. To further define these two research questions a couple of supporting sub-questions are provided, a summary of which is found in Table 1.1.

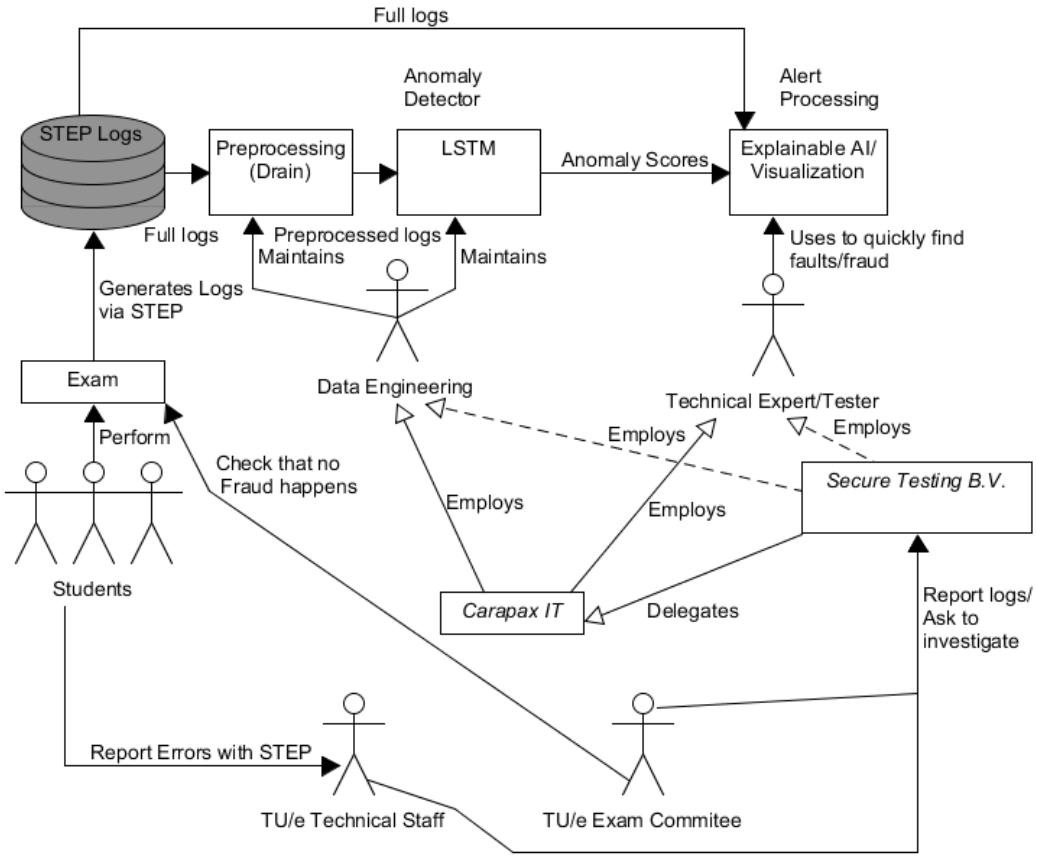


Fig. 1.1: An overview of the fraud analytics framework used and stakeholders.

1. Which methods of anomaly detection and interpretable anomaly alert processing are the most appropriate to implement for anomaly detection for secure testing environments.

To answer this question a literature review will be performed to determine the best algorithms to implement.

2. To what extent does the anomaly detection framework help with the detection of anomalies.

Performance metrics will be calculated for inter-log and intra-log anomaly detection using the available examples of anomalies for intra-log anomaly detection and the different log types i.e. TU/e logs, pentests, fault logs and hackathon logs for inter-log anomaly detection.

3. To what extent does the interpretable anomaly alert processing framework help users gain a better understanding of faults and fraud cases within the system.

A case study will be performed where it will be measured how useful the technical expert and the tester perceive the system.

Research Question	Goal	Method	Data Analysis
1	Find anomaly detection and interpretable anomaly alert processing algorithms that are suitable for monitoring secure testing environments.	Literature review.	Categorize and select algorithms based on their characteristics and how they match the requirements of the business case.
2	Determine whether the anomaly detection framework has good performance.	Choose relevant performance metrics and benchmark on provided/constructed datasets.	Selecting the best performing model based on performance metrics.
3	Determine whether the interpretable anomaly alert processing framework provides value for users.	Perform a user evaluation to gather perceived usefulness and comments.	Analyse the data from this user evaluation session.

Table 1.1: Research questions for this study.

1.3.3 Scope

This study includes only those logs as laid out in section 3.3.1. Two pentest logs have been labeled by the software engineer from Carapax IT, these are the only measures of intra-log anomaly detection that are used. Because only two logs are used and they were labeled by a single person, without anybody else with the knowledge available to verify their labeling, it is

not possible to make a strong case for generalization of the machine learning pipeline as laid out in this study. The results of this study only pertain to the performance of the machine learning pipeline for the STEP system. The usability of the visualization method is only measured by two of the end-users of the visualization at Carapax IT, therefore the visualization is only evaluated within the context of the perceived usability for Carapax IT and the people that are ultimately going to use the visualization and no claims about generalization can be made.

1.3.4 Outline

In Chapter 2 the theory behind this study is explained. The literature review is laid out and preliminary information about algorithms used in this study is given. Techniques for anomaly detection and interpretability are explained and the evaluation metrics that are used in this study are outlined. In Chapter 3 The approach is explained, the data that is used, preprocessing steps, evaluation methods used and interpretability options used. In Chapter 4 the experiments and results for the anomaly detection and interpretability frameworks are shown. Chapter 5 contains the conclusion to the study.

1.4 Contributions

- We defined the problem of anomaly and fraud detection for secure testing environments as a machine learning problem. Different machine learning approaches of the state of the art were defined and applied to the problem. Our thorough search of the relevant literature yielded no other studies exploring this problem domain, thus suggesting that this is the first study to explore the problem domain of anomaly and fraud detection for secure testing environments.
- We proposed Several (modifications of) state of the art deep learning approaches for the secure testing environment domain. In particular, we focused on the LSTM Autoencoder model from (Grover, 2018) in context of the problem and an implementation of the Bi-directional Event Model (BEM) from (Brown et al., 2018) on the problem.
- We conducted a case study of the usability of the visualization in Excel with real logs where it is not known where the real anomalies are. The experts from Carapax IT think the visualization is useful and might help them save time.
- Last, but not least, we turned the lessons learnt from the case study into a guideline to the company how to further develop the machine learning framework for secure testing environments.

Chapter 2

Literature Review

In this chapter a literature review of the anomaly detection domain and interpretability domain is performed. Preliminary information is given about machine learning methods that are used in this work. Further, evaluation metrics used for anomaly detection techniques are outlined and the choices for machine learning approaches made are motivated.

2.1 Anomaly Detection Literature Review

In anomaly detection finding patterns within the data that do not conform to expected behaviour is the main interest (Chandola et al., 2009). In Fig. 2.1, N1, N2 and N3 show examples of areas where within data could be considered as conform normal behaviour. Point A1 shows an example of an anomalous data instance.

2.1.1 Anomaly Detection Challenges

Anomaly detection is a large research field with several sub-domains e.g. (credit card) fraud detection, intrusion detection, fault detection to name but a few. In (Chandola et al., 2009) Chandola, Banerjee and Kumar identify 6 key challenges within the anomaly detection domain.

- It is challenging to define a normal region that encompasses every single instance of normal behaviour. Also observations that lie near the decision boundary can easily be misclassified. An example of this can be seen in Fig. 2.1. Consider the uppermost point of cluster N3, there is a valid case to be made that this point is actually an anomaly as its distance to the cluster center is large and its location is distinct from the other points within the cluster.
- When the anomalous data is generated by a bad faith actor in order to commit fraud or other crimes, then the bad faith actors will try to game the system by generating the anomalous data in such a way that it appears to be conform normal behaviour. This makes it harder to identify anomalies within the fraud and intrusion detection domains, and is of special significance to this study as well.
- Normal behaviour might not be static and evolve over time. This means a model trained now might not be effective a year later.
- The notion of what constitutes an anomaly is different per domain. A definition of an anomaly in one domain might constitute normal behaviour in another. Hence techniques

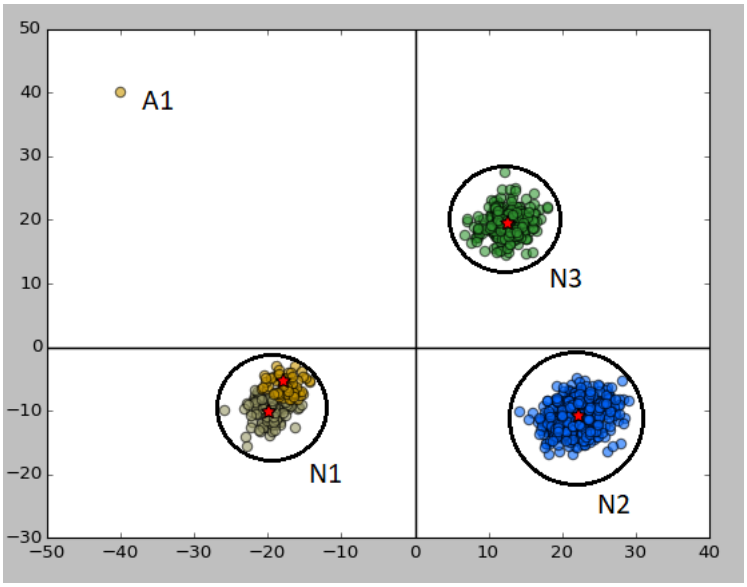


Fig. 2.1: An example of an anomaly in a 2D dataset.

for one domain might not transfer well to another domain. For example a Linux log that contains traces of fraudulent activity, but not faults, can be considered to be an anomalous instance in the fraud detection domain, but a normal instance in the fault detection domain. Even though both are technically anomalous compared to logs that do not contain fraudulent activity and faults.

- In many cases there is no labeled data available because anomalous data is by definition rare and it can be challenging to correctly label anomalous data. Moreover, data that is labeled can be confidential especially in the fraud detection domain. Therefore applying (semi-)supervised learning is often not possible. This makes it harder to build strong models and evaluate them.

- The data might also contain noise that is similar to the anomalies. For example a log-line generated by a verbosity mode can describe normal behaviour but if we do not train on logs using the verbosity mode these log-lines can become noise that will most likely be misidentified as anomalous. It is hard to know whether an instance that is marked as an anomaly is actually a highly valued observation (novelty) or that it is noise that does not have any novelty value. We might not immediately recognize that these log-lines that were incorrectly marked as anomalous are actually describing normal behaviour in a verbose way.

2.1.2 Anomaly Types

In general there are three types of anomalies that are identified.

- *Point Anomalies.* These are specific data instances that are anomalous to the rest of the data. For example A1 in Fig. 2.1 represents a point anomaly. In the context of exam fraud

detection A1 might represent a single log-line which accesses a forbidden website.

- *Collective Anomalies.* These are collections of data instances that as individual data instances might conform to normal behaviour, but as a collective constitute an anomalous sequence. For example consider the regular sequence of a heartbeat in an electrocardiogram, when somebody experiences a heart attack the sequence of the heartbeat changes and this changed sequence is a collective anomaly when compared with the regular heartbeat sequence. In the context of exam fraud detection a collective anomaly might be a large amount of specific actions in a sequence probing for weaknesses in the system.

- *Contextual Anomalies.* These are point anomalies that are only anomalous within a certain given context. For example consider a sensor that measures the temperature every 15 minutes, if the measurements for an hour are [28,29,10,31] then the "10" measurement is anomalous within the context of this sequence, but not on its own. In the context of exam fraud detection a contextual anomaly might be accessing the built-in calculator when it has been disabled for that particular exam.

The anomaly detection challenges as laid out in this section are projected onto the anomaly analytics for secure testing environments domain in Section 2.3.

2.1.3 Anomaly Detection Machine Learning Approaches

Anomaly detection problems deal with limited or no labels and, as a consequence of anomalies generally occurring very infrequently, often have imbalanced datasets where the normal class vastly outnumbers the anomaly class. The goal is to classify data instances/sequences as belonging either to the normal class or the anomaly class, often the data instances get an anomaly score and those scores that are below a given threshold are considered anomalies. There are three types of machine learning that are employed in anomaly detection.

- *Supervised Anomaly Detection.* Using a supervised anomaly detection approach is only possible if the dataset is fully labeled and is therefore often not an option in the anomaly detection domain. Furthermore, it has the additional issue that the datasets are often imbalanced so therefore it requires a specific approach that takes this into account.

- *Semi-Supervised Anomaly Detection.* For this type of anomaly detection methods it is not a requirement that all cases are labeled, in general it's only required that the normal cases are labeled. A model for normal behaviour for the problem domain can then be computed.

- *Unsupervised Anomaly Detection.* For this type of anomaly detection no labels are required at all. Anomalies are found based on only the data instances. The assumption for unsupervised anomaly detection is that the data is imbalanced and normal instances vastly outnumber anomalous ones.

2.2 Interpretability Literature Review

In the interpretability domain finding human understandable explanations of machine learning models and their output is the main interest. Many machine learning algorithms are black boxes which do not tell clearly why a certain output was chosen by the model. To reason about model interpretability a set of definitions which are formalized and agreed upon by experts are needed. In this study the comprehensive taxonomy of the interpretability research domain given by Lipton (Lipton, 2016) will be used.

2.2.1 Transparency

The transparency of a model denotes to what extent the model can be understood. It is the opposite of opacity or blackbox-ness. Transparency can be subdivided into three sub-terms: simulatability, decomposability and algorithmic transparency.

Simulatability

The simulatability of a model is a measure of how transparent the model itself is, simulatability is high if a person can contemplate the entire model at once. A person should be able to take the input of a model and the parameters and in a reasonable time, step through all of the calculations that the model does.

Decomposability

The decomposability of a model is a measure of to what extent each part of the model—each input, parameters and calculation—has a human understandable explanation. For example a decision tree where each node has a plain-text explanation of its role would have high decomposability according to this definition.

Algorithmic Transparency

The algorithmic transparency of a model is a measure of which the learning algorithm used is understandable by humans. For example if it can be proven that an algorithm will always converge regardless of input then it is known that it will still perform on new data. Modern deep learning algorithms however are weak in algorithmic transparency.

2.2.2 Post-hoc Interpretability

The post-hoc interpretability of a model is a measure of the information that can be extracted from a model after training. An advantage of post-hoc interpretability is that already trained models can be understood without having to sacrifice predictive performance by creating a weaker transparent model. Post-hoc interpretability can be subdivided into four sub-terms: text explanations, visualization, local explanations and explanation by example.

Text Explanations

Text explanations in post-hoc interpretability is the usage of natural language text to explain the decisions of a model.

Visualization

Visualization in post-hoc interpretability is the usage of visualization of the model to see if the model has learnt the right things. For example in a convolutional neural network one could visualize the activation of the nodes in a hidden layer for a picture of a dumbbell to see if the model learnt to look at the right features in the image.

Local Explanations

Local explanations in post-hoc interpretability is about trying to explain what a model depends on locally and not globally, for example saliency maps in convolutional neural networks can show which regions of an image produce the most change in output when changed.

Explanation by Example

Explanation by example in post-hoc interpretability is about explaining a decision by giving examples as justification. For example the model could return a certain log-line as anomalous and return the k-nearest neighbours in the model learnt space as explanation for why this log-line is anomalous.

2.3 Interpretable Anomaly Analytics for Secure Testing Environments Definition

A secure testing environment is a custom operating system for the purpose of administering tests/examinations to students where usage of the computer is required but in which certain functions e.g. internet access or access to a calculator are prohibited. Since a custom operating system is used the primary method of determining what happened during a session is by inspecting the logs generated by the custom operating system. We define the interpretable anomaly analytics for secure testing environments domain as follows: It is the intersection between fraud detection, fault detection and jailbreak detection, using log-based anomaly analytics methods for secure testing environments. The jailbreak detection for secure testing environments is novel to this work to our knowledge. Figure 2.2 contrasts jailbreak detection with intruder detection. In intruder detection, there is a secure environment which is locked and bad faith actors are attempting to circumvent the lock and enter the secure environment. In jailbreak detection there is a secure environment which is locked, however the bad faith actors are locked inside the secure environment and their goal is to circumvent the lock to exit the secure environment while still having access to the privileges provided by the secure environment. It is also possible for a bad faith actor to first intrude into the system to change it in preparation for a later attempt at a jailbreak attack. The anomaly detection challenges as laid out in Section 2.1.1 relate to the anomaly analytics for secure testing environments domain as follows.

- Defining a normal region that encompasses every single instance of normal behaviour is also challenging within the anomaly analytics for secure testing environments domain. Log-lines and logs can similarly be plotted according to their anomaly scores, this leads to the same issue that normal behaviour might not be easily defined by a normal region.
- In the anomaly analytics for secure testing environments domain anomalous data that is purposely generated by bad faith actors to avoid detection is also a real problem. Generating log-lines in such a way that related log-lines are not next to each other can make anomaly detection by collective or contextual anomalies harder. Also generating a lot of

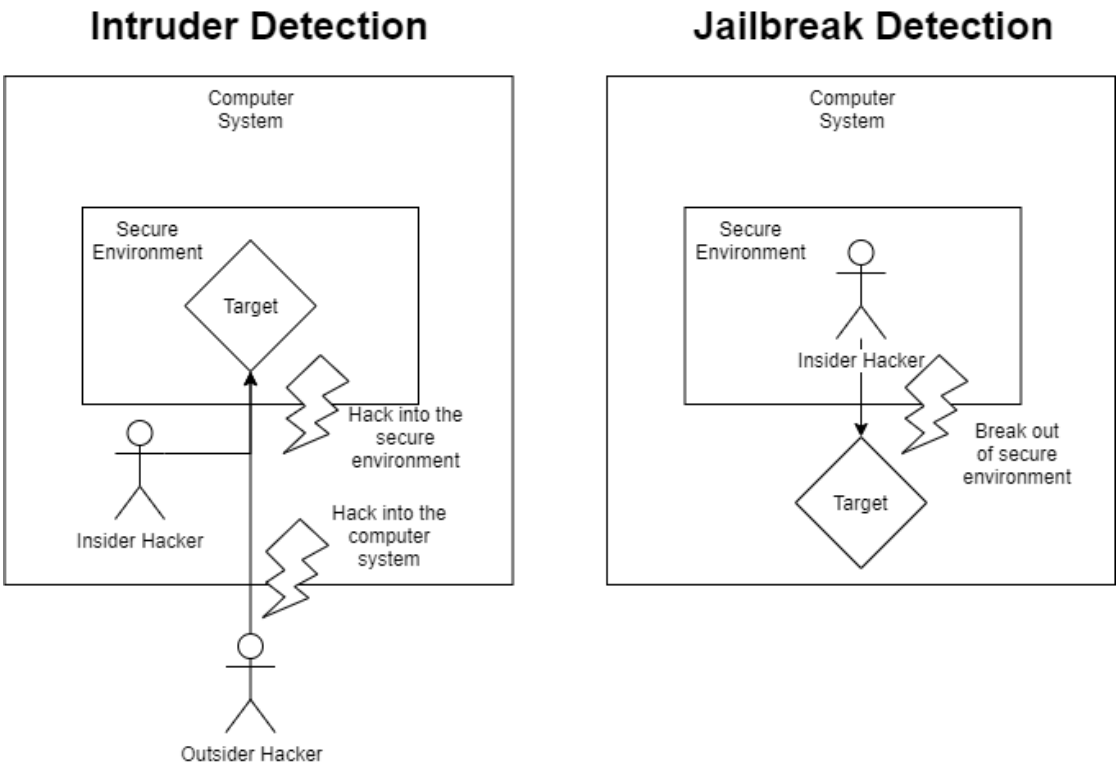


Fig. 2.2: Contrasting intruder detection with jailbreak detection.

(unique) normal log-lines might allow a log with only a couple of malicious log-lines to fall through the cracks, especially when using a anomaly score that uses the mean over all log-line anomaly scores.

- In the anomaly analytics for secure testing environments domain, normal behaviour also changes over time. As the environment is updated over time with new capabilities, it is highly likely that normal behaviour will change over time.
- In the anomaly analytics for secure testing environments domain, the notion of what constitutes an anomaly can be different things. In this work it is either a point anomaly in a single log-line or a point anomaly between full logs. Collective and contextual anomalies between log-lines are also possible within this domain.
- In the anomaly analytics for secure testing environments domain, it is also the case that there often is no labeled data available. Logs that are known to include fraud are needed for this. However, since fraud on exams is relatively rare it would be hard to know which logs contain fraud, unless you generate fraud cases by letting professional hackers loose in the environment. Then the question will still be whether fraud generated by professionals is visible in logs in a similar way as fraud generated by a student during a real exam.
- In the anomaly analytics for secure testing environments domain, noise can also be similar to anomalies. Noise in logs can be described as log-lines that are irrelevant but rare, then they are technically anomalous but not novel and useless for finding fraudulent logs. As well as logs that are highly different from the norm, but not anomalous, for example when log-lines from a log are generated using a verbosity setting.

2.4 Characterization of Machine Learning Approaches for Anomaly Detection and Choices

The goal of this section is to explain all the background information that is required in order to be able to answer the first research question: *"Which methods of anomaly detection and interpretable anomaly alert processing are the most appropriate to implement for anomaly detection for secure testing environments."*

2.4.1 Machine Learning Approaches for Anomaly Detection

Introduction

In order to decide which machine learning model is most appropriate to implement for the anomaly detector from the use case in Fig. 1.1 the data and solution space of the problem is analysed. From the amount of logs that are received in a live environment, extremely few will contain fraud attempts, some will have faults, but this is rare as laptops are tested by the TU/e in advance to make sure STEP runs on the laptops. This means the problem has an imbalanced nature as there are many more normal logs than anomalous logs. If a log is anomalous then within the log, the anomalous lines might not be very imbalanced compared to the normal lines. The anomalous lines are mostly imbalanced with regards to the total distribution of log-lines over all logs received, this is a property that can be exploited in order to classify anomalous lines using one-class classification on normal log-lines. An important choice to make is whether a generative or a discriminative model will be used and whether on-line or off-line learning should be used.

Imbalanced Classification

The dictionary definition of an anomaly is "something that deviates from what is standard, normal, or expected." [1]. By definition therefore, the search is for specific behaviour or instances in the data which are not standard and thus rare. This leads to the primary characteristic and problem of anomaly detection, dealing with imbalanced datasets where the number of normal instances or behaviour is much higher than the number of abnormal instances. This makes anomaly detection challenging with traditional classification algorithms if the imbalanced nature of the data that one is interested in classifying is not explicitly accounted for e.g. if there is a dataset with 1000 instances of which 10 are anomalies a classification algorithm might say "classify every instance as normal" and this algorithm will have an accuracy of 99% whilst having absolutely no utility for finding the anomalies one is interested in finding. This means some special strategies need to be used to apply classification for anomalies, for example false negatives can be weighted down aggressively such that the model will be more likely to generate false positives in order to not misclassify the anomalies.

One-Class Classification

In one-class classification the goal is to create a model recognizing objects from a single class using data containing only that single class (Tax, 2001). Once a model has been created that can recognize a single class, anomalies are defined as any object that does not belong to this class. For example in anomaly detection one-class classification is used to classify only normal instances. Then this model can be used to classify everything that does not fall under the normal class as anomalous.

Statistical Outlier Detection

There are also statistical definitions of outliers, for example the Z-score can be used for finding outliers in Gaussian distributed data. The Z-score for a single data instance x is given by $z = \frac{x-\mu}{\sigma}$ for population mean μ and population standard deviation σ and computes the number of standard deviations a data instance is away from the data mean. A threshold value can be compared with the absolute Z-score of a data instance to determine if this instance is an outlier.

Supervised vs Unsupervised Learning

Two kinds of learning methodologies are considered, supervised and unsupervised learning. For supervised learning every data instance needs to have a label, the goal of a supervised learning algorithm is to learn a model which tells which label to give to a new data instance. In unsupervised learning data instances do not have a label, the model has to infer the different classes of the data instances itself.

Generative vs Discriminative Models

There are two kinds of statistical outlier models, generative and discriminative. A generative model models the actual distribution of a class i.e. $P(x, y)$. A discriminative model models only the decision boundary between classes i.e. $P(y|x)$. A generative model works well with unsupervised learning, but requires a lot of data. A discriminative model requires less data and is less computationally expensive but does not work well with unsupervised learning. Examples of generative models are naïve bayes and hidden markov models, examples of discriminative models are nearest neighbour models and logistic regression.

[1] https://www.lexico.com/en/definition/anomaly

Engineered Features vs Raw Data

Two kinds of available data are considered. Data which has engineered features, usually extracted and constructed manually from the data by an expert and raw data which does not have features. The mathematical space created by the data features is called the feature space. There are machine learning algorithms that can work directly on the feature space like nearest neighbour. Not in all cases it is desirable to manually craft the features from the data, for example if the data is highly complicated or not understandable for humans. In fact it can be argued that an important goal for artificial intelligence is to remove the need for manual feature engineering. In the case of raw data, where it's not possible or desirable to engineer features, a machine learning algorithm is required that can work directly on the raw data and automatically extract the features it needs. While engineered features can be really powerful it is also possible for specific machine learning algorithms to learn features from raw data which match or exceed the features created by an expert. Whether it is better to use engineered features or raw data depends on the data. If the data is from a domain that is known to use machine learning algorithms that perform best on raw data, e.g. image recognition, or where features are hard to engineer manually then using the raw data is the way to go. If manual feature engineering outperforms algorithms that automatically extract features or if human understandable features are required for other reasons then using manual feature engineering is the way to go.

Deep Learning vs Traditional Machine Learning

Both deep learning and traditional machine learning algorithms are machine learning algorithms. The advantage of deep learning over traditional machine learning is that deep learning models can also extract features from raw data, negating the need for feature extraction. Deep learning models are therefore useful for dealing with problems that are complicated to solve with traditional machine learning or have complicated feature engineering steps. Examples of problems that deep learning is better fit to solve than traditional machine learning are natural language processing (NLP) and image recognition. Deep learning also shines when there is a lot of data available, but in cases where less data is available traditional machine learning algorithms will often outperform deep learning methods.

On-line vs Off-line Algorithms

An on-line algorithm is an algorithm that can receive a sequence of requests and performs an immediate action to each request (Karp, 1992). In contrast, off-line algorithms cannot process a sequence of requests because they require full knowledge of whatever it is they are processing. While off-line algorithms often perform equally well or better than on-line algorithms, their requirement to know the sequence makes it infeasible in situations where the sequence is very long or even theoretically infinite. In machine learning, on-line algorithms are those that update the predictor after being fed each data instance in a sequence, while off-line algorithms—more often called batch learning for machine learning problems—learn on the entire training data at once. Batch learning can be infeasible if the training data is very large.

Motivation of Choices

The data is imbalanced and since there are not a lot of labels for anomalous data instances it makes more sense to use an unsupervised learning algorithm rather than a supervised learning algorithm. Then because it was decided to use an unsupervised learning algorithm, creating a generative model is preferred over a discriminative model because generative models work well

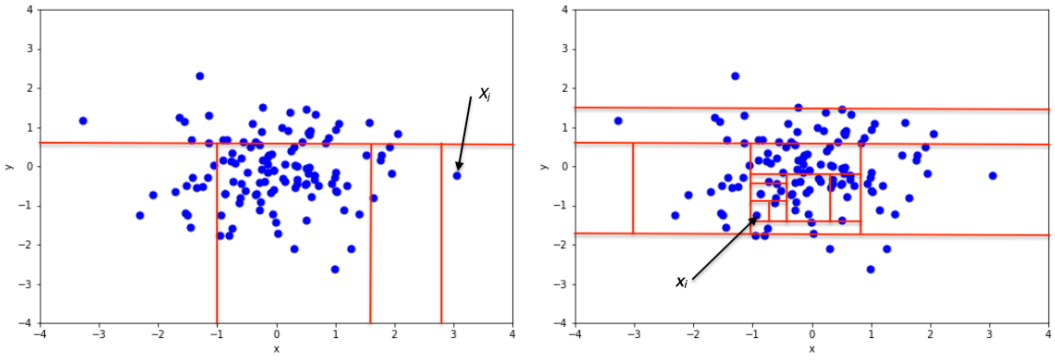
with unsupervised learning and discriminative models do not. Moreover, there is a lot of data available so the large amount of data needed for a generative model should not be a problem. Because the data is very complicated and does not have clear features, it seems a reasonable idea to use raw data and try to create a deep learning model over a traditional machine learning model. The data being lines of text would be a good fit for an NLP model which works better with deep learning and there is enough data available to make a deep learning model feasible. Another feasible approach would be a model using autoencoders. While deep-learning seems to be the best fit, there are several unsupervised traditional machine learning methods which could be useful and will be explored like an isolation forest and clustering using similarity measures that work with complicated data. Since the model is allowed to be created during off-hours and there are no time constraints, there is no specific reason to use on-line learning over off-line learning. The data can be quite large but it is not large enough to warrant using on-line learning at this moment. Another reason to not use on-line learning is because on-line learning is more challenging than off-line learning due to added time-constraints and the constraint of the model looking at examples only once as they are streaming in. From a practical applicability perspective, it is interesting to study both off-line and on-line learning. However, it was decided to only look at off-line learning because, given the characteristics of this domain, the problem is already expected to be highly challenging.

2.5 Preliminary Overview of Machine Learning Approaches Used

2.5.1 Isolation Forest

Isolation forest (Liu et al., 2008) (Liu et al., 2012) is an anomaly detection method whose goal it is to find anomalies by isolating them. While most anomaly detection methods work by creating a profile of normal instances and comparing new instances to that profile, isolation forest in comparison does not create a profile but searches for anomalies exclusively. The advantages of the isolation forest compared with profile based models are that the amount of false positives are reduced because profile based models are optimised for creating a profile of normal instances rather than finding anomalies and that isolation forest has a linear computational complexity, which makes it faster than most methods.

Isolation forest works by creating a forest of k isolation trees with a height limit of l. Each isolation tree is built on a sub-sample of ψ instances from input data X, the height limit is defined as $l = \lceil \log_2 \psi \rceil$, this is approximately the average tree length (Knuth, 1998). The isolation tree is created by selecting a random attribute and selecting a random split value between the min and max values of this attribute. Based on this split value the sub-sampled data is divided into two smaller isolation trees which are recursed on until there are less than 2 instances in the division (isolation) or the height limit l is reached. The idea behind the isolation forest is that anomalous instances require on average less splits to isolate than normal instances, this is illustrated in Fig. 2.3. A sub-sample is used because the algorithm works better on a smaller set of data, as the amount of instances increases it becomes harder to properly isolate individual instances. The reason that the height limit is set to approximately the average tree length is because only short path lengths are interesting because they denote anomalous instances. The anomaly scores of individual instances can then be found by running them through the forest of isolation trees and calculating the expected path length for each instance.



(a) Isolating an anomalous instance

(b) Isolating a normal instance

Fig. 2.3: An example of an isolation tree isolating an anomalous instance in Figure (a) and a normal instance in Figure (b).
Images by Sal Borrelli [CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0)]

2.5.2 Long Short-Term Memory

Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) (LSTM) is a type of recurrent neural network (RNN) created to solve the problem of vanishing gradients in regular RNNs. In a basic RNN—such as the one shown in Fig. 2.4—the state of the network is retained and can be used as a sort of memory for earlier input in a sequence of inputs. In theory this would allow the model to use context from the past in order to give better predictions. In practice however the gradients tend to "vanish" (i.e. getting a value close to 0) exponentially over time when there is a big temporal delay between big events in the network. LSTMs were created as a solution to this problem. LSTMs have a cell state (denoted by c_t for a given timestep t in Fig. 2.5) which flows through every recursion of the cell and contains information from the past which the model has decided is important to remember. The cell state is regulated by a forget gate with a sigmoid layer—denoted by F_t—a sigmoid layer returns values in the range [0,1], hence when multiplied with the cell state C_t it regulates how much of the previous cell state C_{t-1} to keep. Next there is the input gate layer denoted by I_t, the sigmoid layer decides which values to update and the tanh layer selects the candidate values to add to the state, these two layers are then multiplied and added to the state. Finally, in the output layer O_t a sigmoid layer decides what values of the cell state that are going to be output. The LSTM allows to remember long-term dependencies while also more carefully regulating the gradient decay preventing the exponential gradient decay of the vanishing gradient problem.

2.5.3 Autoencoders

Autoencoders, as shown in Fig. 2.6, are a type of neural network configuration that encodes its input to a latent space and then decodes/reconstructs a learned representation of the input from the latent space (Kramer, 1991). Autoencoders can be used for dimensional reduction of the input. Of note to the anomaly detection problem is their recent application as a generative model. If the latent space has learned a good representation of the input space it can generate its own output from the latent space. For anomaly detection, if an autoencoder is trained on only normal data, then when anomalous data is encoded the autoencoder will not be able to capture the details of the anomalous data correctly in the latent space leading to reconstruction

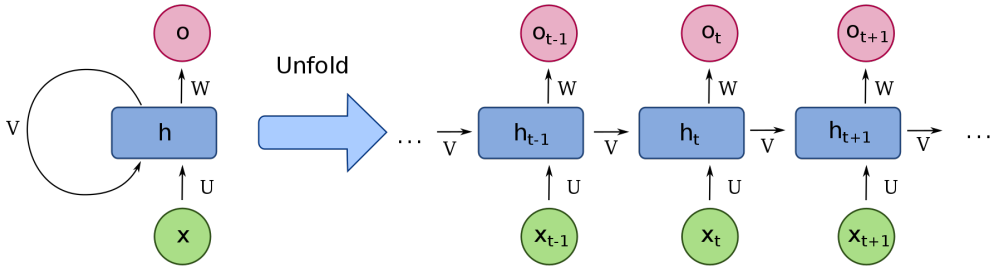


Fig. 2.4: An example of a basic recurrent neural network.
Image by François Deloche [CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0)]

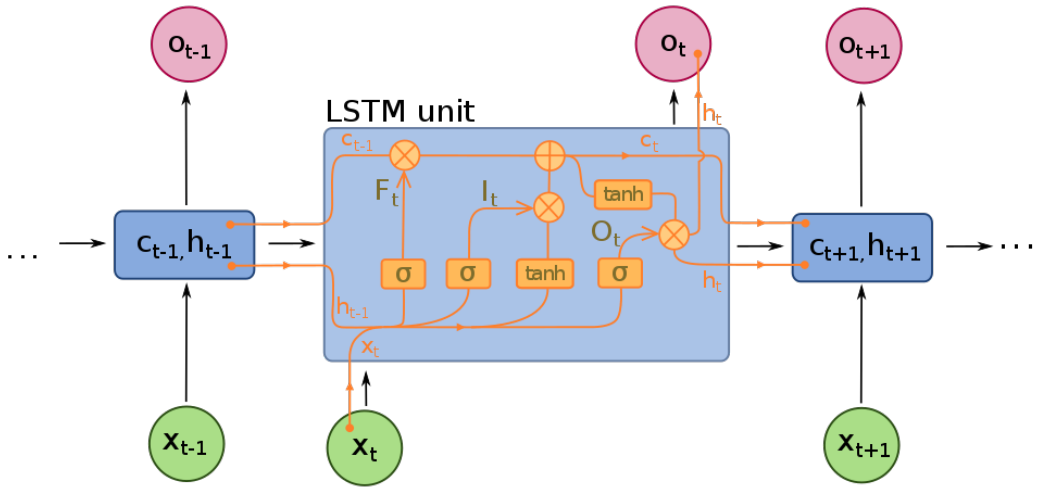


Fig. 2.5: An example of a LSTM.
Image by François Deloche [CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0)]

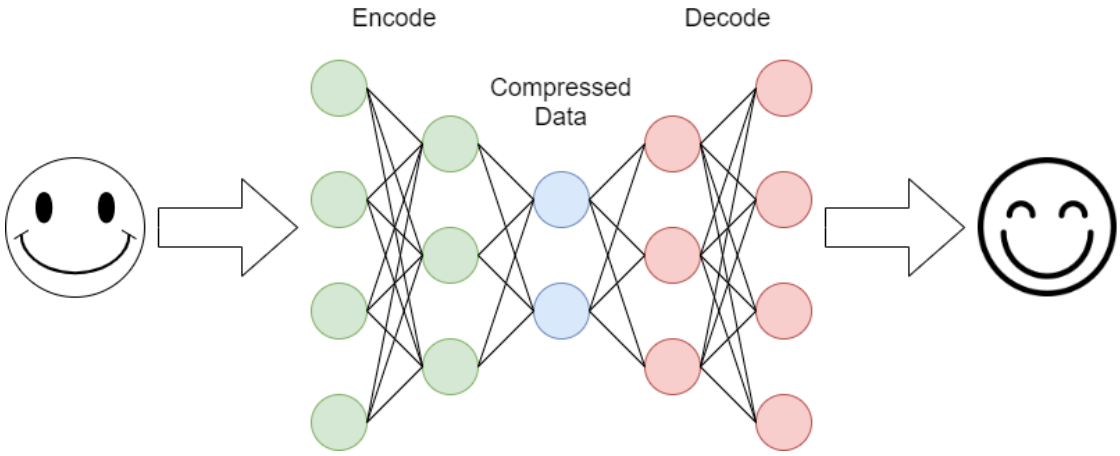


Fig. 2.6: An example of an autoencoder.

errors after the decoding step. The measure of reconstruction error for an input can be used as an anomaly score for that input.

2.5.4 Attention

Attention mechanisms are a way to model dependencies within the input of a deep learning model without regard for the distance between the values in the input. This is important because regular RNNs forget states that came much earlier in a sequence. For example, in the input sentence "The man picks up the ball, looks around, pauses, and throws it at the wall." an attention mechanism would allow the model to learn the dependency between "ball" and "it". Rather than adding attention within the RNN layer, the attention mechanisms were captured in a single deep learning layer that can be used on its own (Vaswani et al., 2017). The attention layer is trained to learn which states to pay more attention to when predicting the next token in a sequence. For example when predicting "it" in the previous example the attention layer will give a high weight to the word "ball" and a low weight to the other words.

2.6 Preliminary Overview of Evaluation Metrics for Anomaly Detection Techniques

The evaluation metrics that are used are outlined and explained in this section.

2.6.1 Precision and Recall

When performing binary classification there are four categories to which the predictions can belong. Each element has a true value, either negative or positive, and after classification, each element will have a predicted value of either negative or positive. When the predicted value is correct with regards to the true value there are the following two categories: negative values that were predicted as negative are called true negatives (TN), likewise positive values that were predicted as positive are called true positives (TP). When the predicted value is mistaken with regards to the true value, there are the following two categories. Positive values that were incorrectly predicted as negative are called false negatives (FN) and negative values that were

incorrectly predicted as positive are called false positives (FP). The distribution of the elements into these four categories is often presented in the form of a confusion matrix. An example of a confusion matrix is given in Table 2.1.

Confusion Matrix	Predicted Values		
True Values		Negative	Positive
	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

Table 2.1: The definition of a confusion matrix

From the confusion matrix several model performance metrics can be derived. Precision is a measure of how many of the positive predicted elements are truly positive, it is given by: $precision = \frac{TP}{TP+FP}$. Recall is a measure of how many of the elements that are truly true were predicted to be true, i.e. how many of the relevant elements have been found by the algorithm, in the case of the anomaly detection problem it is how many anomalies have been found versus the ones that are not found by the model. Recall is defined by: $Recall = \frac{TP}{TP+FN}$. Visually an example of precision and recall is offered in Fig. 2.7.

2.6.2 F1 Score

The F1 Score is defined as the harmonic mean of the precision and recall metrics.

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

It is a stronger measure of performance than the precision and recall alone, as it combines precision and recall to give a single score. The F1 Score is useful in anomaly detection because often the precision and recall alone give a skewed view of the performance of the model. For example, if the model has high precision but low recall, then most of its predictions are correct but the model is missing a lot of anomalies. If the model has high recall but low precision then the model predictions are mostly wrong, but it does find most of the anomalies. It is of course very bad for an anomaly detection model if it would predict 80% of the data elements to be anomalous, even if 100% of the actual anomalies are within this set. Keeping false positives low is especially important in anomaly detection, as datasets contain many more false elements than true elements. The case where precision is high but recall is low is preferable to the case where recall is high but precision is low, in the best case however, both recall and precision should be high. As the F1 computes the harmonic mean of precision and recall it punishes models which excel in either high precision or high recall but not in the other metric. Hence the F1-Score is a better metric than only the precision or recall for anomaly detection. However, since neither the precision nor recall take into account the imbalanced nature of the dataset in anomaly detection the F1-score does not either.

2.6.3 Accuracy

Accuracy is defined as the total amount of correct predictions over all predictions:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

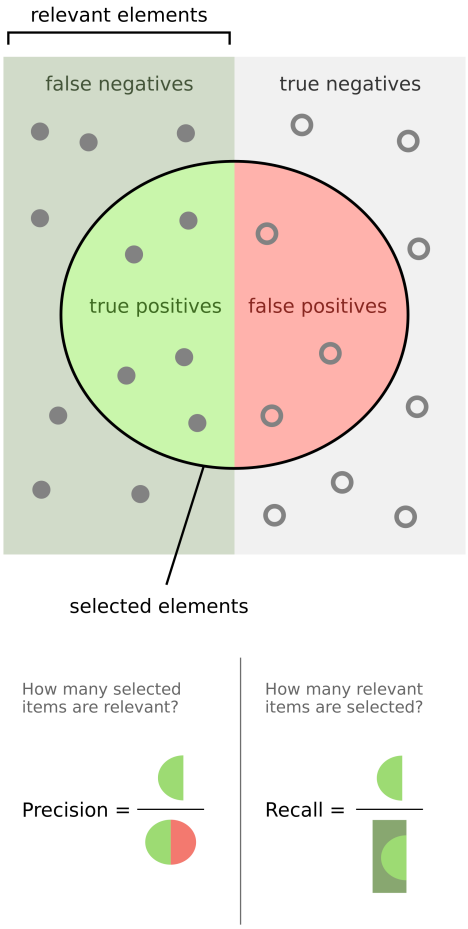


Fig. 2.7: An example how precision and recall are defined visually. Image by Walber [CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0)]

Balanced Accuracy

In imbalanced datasets the accuracy is often not as useful because the majority class will dominate the accuracy score. Often the minority class is more of interest in imbalanced datasets so the balanced accuracy measure is more useful in those cases, as positives and negatives are balanced to be equal even if the class sizes are highly imbalanced. Balanced accuracy is given by:

$$BACC = (\frac{TP}{TP + FP} + \frac{TN}{TN + FN})/2$$

The difference between the balanced accuracy and F1 Score is that the F1 Score does not take into account the amount of true negative instances at all and the amount of total negative instances is irrelevant. The balanced accuracy in comparison does take into account the negative instances, but balances it out by giving half of its weight to the percentage of true positives found and half of its weight to the percentage of true negatives found. The F1 Score can be more useful than the balanced accuracy in the case where you have an imbalanced dataset and you care about finding true positives, but (the amount of) true negatives are unimportant.

2.6.4 Area Under the Receiver Operating Characteristic Curve

The Area Under the Receiver Operating Characteristic curve (AUROC or more commonly AUC (Area Under the Curve)) is one of the most commonly used metrics for machine learning models.

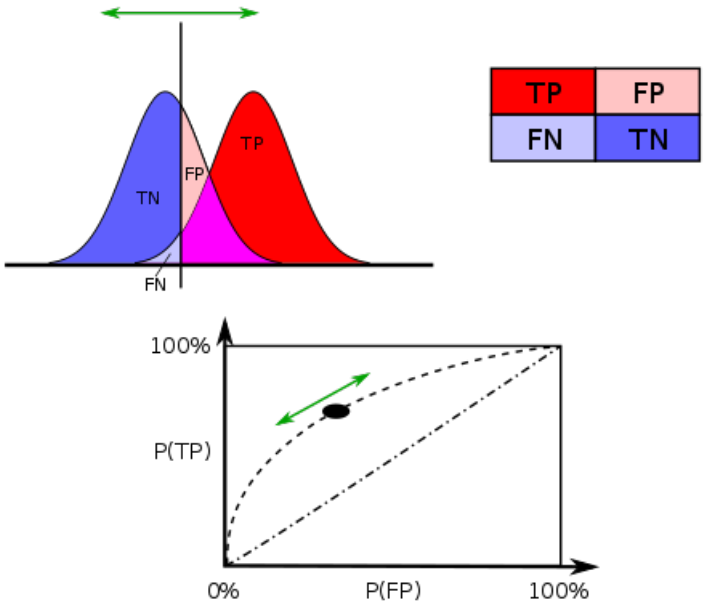


Fig. 2.8: An example of how the ROC curve is constructed. Image by kakau based on original by Sharpr [CC BY-SA (http://creativecommons.org/licenses/by-sa/3.0/)]

The Receiver Operating Characteristics (ROC) graph is created by plotting the *tp rate* against the *fp rate* (Fawcett, 2006). The *tp rate* is the same as the recall and the *fp rate* is the false alarm rate (what percentage of negative instances is given as false positives). They are given by:

$$tp\ rate = \frac{TP}{TP + FN}$$

and

$$fp\ rate = \frac{FP}{FP + TN}$$

The ROC curve is created by varying a threshold value which decides which instances are positive or negative based on their probability being positive according to a model. As the threshold is varied, as shown in Fig 2.8 the trade-off between *fp rate* and *tp rate* is plotted, the point at 0% *fp rate* and *tp rate* represents the case where all instances are classified as negative, no false positives are given but no true positives either. In contrast, the point at 100% *fp rate* and *tp rate* represents the case in which all instances are classified as positives, finding all true positives but also giving every negative instances as a false positive. The ROC graph thus represents the trade-off between *fp rate* and *tp rate*, it shows the price you pay in the amount of false positives for a specific detection rate/recall. By plotting the *fp rate* versus the *tp rate* a curve in created in ROC space, the AUC is then calculated by taking the area under this curve giving a single valued metric summarizing the performance of the machine learning model.

It is important to note that the minimal useful value the AUC can take is 0.5 as, if the AUC would take a value below 0.5 one could simply reverse the classification of positives and negatives by the algorithm to get an AUC score of $1 - x$ for some $x \leq 0.5$ which is then always larger or equal to 0.5.

The AUC is considered to be superior to accuracy as a performance metric, as there has been a realization that accuracy is often a poor metric for measuring performance (Provost et al.).

Partial AUC

For imbalanced data it can often be useful to compute only the AUC of a partial section of the ROC graph (McClish, 1989). For example, this can be the case if it is known that cases with a *fp rate* > 0.2 are not useful. In those cases the partial AUC (pAUC) can be computed by taking the area of the ROC graph up to a *fp rate* of 0.2. However, the pAUC cannot be fully interpreted as the same metric as the AUC in this way because information about actual negatives is removed (Carrington et al., 2019).

2.6.5 Precision-Recall Curve

Comparable with the ROC graph, the precision-recall graph plots the model precision against the model recall. The precision-recall curve is plot in the same way as the ROC curve by varying a threshold value over the probability of instances being positive. It shows the trade-off between precision and recall and the price you pay in lower precision for higher recall or vice-versa. The precision-recall curve can be a better metric than the ROC curve for imbalanced data (Saito and Rehmsmeier, 2015).

Average Precision

Like the AUC is equal to the area under the ROC curve, the average precision (AP) of a model is defined as the area under the precision-recall curve (Su et al., 2015).

2.7 Design Choices

The literature of the anomaly detection domain and interpretability domain was reviewed and a definition of the anomaly analytics for secure testing environments domain was defined. Next,

different machine learning approaches for anomaly detection were discussed. The following design choices were made. Because the dataset is imbalanced with regards to anomalies and labels are missing, it was decided that unsupervised learning algorithms look the most promising. Because of this it was decided that a generative model is the preferred model type for the problem. The data is complicated and does not have clear features so a deep learning model was chosen as the most promising kind of model to look into. Because there are no execution time restraints from the company and the amount of data is not too large, it was decided that on-line learning was not necessary and off-line learning will be used. A model using an autoencoder or NLP seems to be a natural choice to look into further. Next an overview of machine learning approaches and evaluation metrics is given.

Chapter 3

Deep Learning Approach for Anomaly Detection in Secure Testing Environments

In this work the cross-industry standard process for data mining (CRISP-DM) (Shearer, 2000) is adopted. CRISP-DM provides a clear helicopter overview of the different phases in a data mining project. Our approach is framed within CRISP-DMs distinct phases to provide a clearer overview of the approach.

3.1 General Model Development Approach

CRISP-DM consists of 6 phases as shown in Fig. 3.1:

1. **Business Understanding:** The first phase of CRISP-DM is to understand the needs of the customer and converting their needs to a data mining problem definition and a project plan. It is important to understand the true needs of the customer such that there will not be any mistakes in the end, like finding the right answers to the wrong question. A measure of evaluation should be established which determines whether or not a model is successful in the eyes of the customer. The goal of the model should be established in business terms. Moreover it is important to assess which data is available.

2. **Data Understanding:** The second phase of CRISP-DM is to understand the data. The data will be collected and a description of the data will be made. The important question in this phase is: "Does the data acquired satisfy the relevant requirements?". Then the data will be explored and any data mining questions that can be answered already with querying, visualization and reporting are answered. Finally the quality of the data will be verified by noting whether the data is complete or if there are strange attributes in the data that conflict with common sense.

3. **Data Preparation:** The third phase of CRISP-DM is the preparation of the final data set from the raw data. First the data that will be used is selected and an argumentation for the choices made for the selected data is given. Second the selected data is cleaned and issues with the data quality as outlined in the data understanding phase are addressed. Once the data is cleaned the data is prepared by creating new records or producing derived

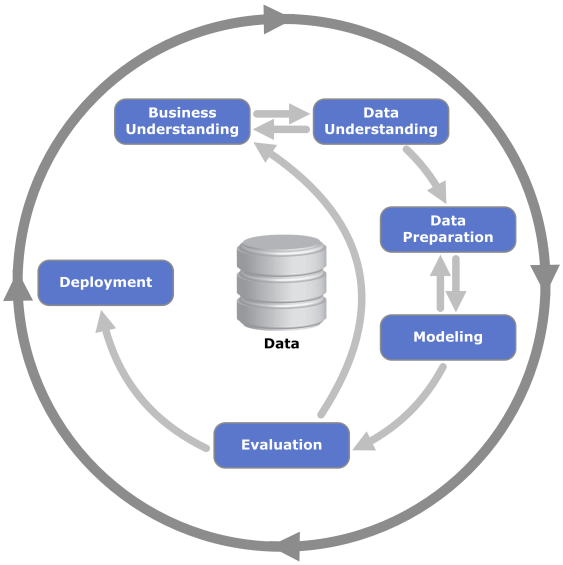


Fig. 3.1: The cross-industry standard process for data mining (CRISP-DM). Image by Kenneth Jensen [CC BY-SA (https://creativecommons.org/licenses/by-sa/3.0)]

attributes. Then the data is integrated by combining data from multiple tables or records into new records. Data integration also includes aggregation of data. Finally the data is formatted by changing the format or design of the data. This includes the removal of stop words, trimming text and other formatting steps. This is an especially important step for the STEP problem especially if NLP is used to address it.

4. **Modeling:** The fourth phase of CRISP-DM is the modeling phase. In this phase the modeling techniques are selected and applied and parameters of those techniques are set. First the modeling techniques and algorithms are selected. Second the quality and validity of the model is determined by empirical testing. This includes splitting the data into a train and test set to estimate the quality of the model by measuring the error rate on the test set. Then the model is built and assessed.

5. **Evaluation:** The fifth phase of CRISP-DM is the evaluation of the model in terms of the achievement of business objectives. First the results are evaluated and it is checked whether the model still has some business deficit. If there is time and budget the model can also be tested in real-world applications. Next the model itself is reviewed to see if anything has been missed or overlooked. Then quality assurance steps will be triggered. Finally it is decided whether the model is ready for deployment or that it still needs revisions, in this case the model development returns back to the first phase of CRISP-DM business understanding.

6. **Deployment:** In the final phase of CRISP-DM the model will be handed over to the customer who will deploy it into their decision making processes. It is important that the customer is informed about all the details of the model and knows how to use the model in their processes. The documentation of the model into a report can be part of this phase, as well as reflection on the process. The deployment phase is not explicitly part of this thesis, the focus is on the first five phases of CRISP-DM.

3.2 Business Understanding

The business understanding is outlined in sections 1.1 and 1.2. The goal of the model in business terms is outlined in subsection 1.2.5.

3.2.1 Main Business Goals

- The development of a solution to find fraud and faults in STEP sessions.
- The development of a presentation of the results that decreases the time experts require to find fraud and faults in STEP sessions.

3.2.2 Secondary Business Goals

- An implementation of the solution incorporated into STEP.
- The development of a solution to automatically find STEP sessions that contain a relevant amount of fraud or faults.
- The presentation of the results is made understandable for lay people.

3.2.3 Business Goal Measure of Evaluation

There is no explicit measure of success for the project. The goal is to find the best possible solution to the problem within the allotted time frame. The evaluation of success will happen outside of the scope of this work.

3.3 Data Understanding

An overview of the collected data is given and the preliminary choices made in regards to what data is being used are explained.

3.3.1 Available Data

The available data comes in part from the Linux distribution, which are either standard Linux logs or custom logging. Furthermore, there is data available through other collected logs. The complete set of the packages of logs will be referred to as a session. There are 3639 sessions generated during exams at the TU/e and used with permission. The assumption is made that these logs only contain non-anomalous data and these logs will be used as examples of normal data. It is unknown whether this is actually the case, as it's infeasible and prone to mistakes to manually confirm this for each package of logs. Henceforward these 3639 sessions shall be called the TU/e dataset. There also is a set of 27 sessions which were generated by a pentest company while testing the security of the STEP system. A pentest stands for "penetration test" and is a form of ethical hacking where hackers attempt to break into a computer system in order to test the security of said system. While the pentest company was not able to breach the security of the STEP system, it is assumed these sessions contain examples of anomalous data, however it cannot be confirmed with absolute certainty that all 27 sessions contain hack attempts. Henceforward these 27 sessions shall be called the pentest dataset. Furthermore, there is a set of 41 sessions which contains faults that were reported in order to improve the STEP system. Because these sessions were sent in because of errors and they were checked by Carapax IT, there is a high

degree of confidence that these logs actually contain anomalies. A disadvantage of this dataset is that the sessions are from different versions of the STEP system, sessions might contain unique log-lines depending on the version of the STEP system which generated it. Henceforward these 41 sessions shall be called the fault dataset. In addition to this, not all 41 fault sessions were available from the start. The older version of the fault dataset is called the partial fault dataset and contains 13 sessions which are also still part of the full fault dataset. It is important to note that the 13 sessions from the partial fault dataset are older than the 28 sessions that were added for the full fault dataset. Finally there is a set of 37 sessions which contain hack attempts by students during a hackathon organized by Secure Testing B.V. at the TU/e. As with the pentest dataset it is assumed that these 37 sessions contain anomalous behaviour. However, it cannot be confirmed with absolute certainty that all 37 of these sessions actually contain hack attempts. Henceforward these 37 sessions shall be called the hackathon dataset. Not all datasets were available at the start of the project, the fault dataset and hackathon dataset were added several months after the start of the project. Hence, the results in chapter 4 show the evolution of experimentation as new data became available. Table 3.1 gives an overview of the available sessions.

Dataset List				
Dataset Name	Overview	Number of Sessions	Datatype	Limitations
TU/e Dataset	Sessions generated during real TU/e exams.	3639	Non-Anomalous	No complete confidence that every single session is non-anomalous.
Pentest Dataset	Sessions generated during pentests.	27	Anomalous (Hack)	No certainty that every session contains hack attempts.
Fault Dataset	Sessions that contain known faults.	41	Anomalous (Fault)	Many sessions are from different versions of the STEP system
Hackathon Dataset	Sessions generated during TU/e hackathon.	37	Anomalous (Hack)	No certainty that every session contains hack attempts.

Table 3.1: Overview of different datasets.

3.3.2 Data Choices

Because the syslog includes an overview of many different processes in the STEP system and to avoid over complicating the data extraction it was decided to focus on using the syslog only. The technical expert of Carapax IT was asked to label all log-lines that contain anomalies he could find in two syslogs of the pentest dataset. These two labeled syslogs are the primary method of evaluation for the intra-log anomaly detection models. We use these labels to evaluate our predictions for anomalies on log-line level (intra-log anomaly detection). And we can use many log-line predictions/scores to predict anomalies on log level which can be evaluated by checking which dataset our predictions belong to (inter-log anomaly detection). In a language model we can also predict anomalies on character and word level, however these cannot be evaluated at that level.

Limitations of Data Choice

The decision to only use the syslog has an important limitation in that the syslog does not contain log-lines about every single fault and hack attempt possible in the system. Hence there are certain faults or fraud attempts that cannot be found using only the syslog. Because there are only two labelled datasets which have been manually labelled by the technical expert of Carapax IT there is currently no very strong evaluation framework. Furthermore, it cannot be assured that every anomaly that is in the dataset has been found. The assumption is made that the pentest dataset contains logs that are similar to real fraudulent student logs. The same is true of the hackathon dataset. An open question is to what extent these datasets align with actual fraudulent student logs. Furthermore, since it cannot be assumed that the pentests which were labelled have a complete labelling with no mistakes, actual fraudulent student logs would have to be seen to know for sure if the model can find them. It is possible that the pentest datasets are just fundamentally different to the TU/e datasets because of changes to the system that were made after the TU/e datasets were generated. Another consideration is that a part of the pentest logs appear to have some kind of verbosity mode on, which means they have a lot of very in-depth log-lines that do not appear in any of the logs of the TU/e dataset but are not necessarily anomalous. Therefore this reduces the usability of the pentest logs as any generative model that uses TU/e logs as a training set will potentially have a lot of false positives on the pentest logs with verbosity mode on. The question is whether these logs are still of any use with the verbosity mode on. Another limitation of the syslog is that the space of potential log-lines is very large and even infinite in theory. This is because log-lines can be added by the STEP system and the Linux distribution over time. Also, if the code that defines the log-line has a variable in it, any log-line created by this code can be unique. This can be an issue for machine learning methods that rely on sets of unique events for example, as well as for methods that use vocabularies of words as it can explode the size of the vocabulary with unique words that are low quality, which can drown out actual rare words that capture anomalous behaviour.

3.4 Data Preparation

To address the limitation of the syslog containing too many unique events a method is needed to reduce the amount of unique events that exist within the syslog. Unique events can be reduced by using a log key extraction algorithm which attempts to remove variables from log-lines so log-lines with variables in them no longer generate infinite events. The two pentest syslogs that were labelled are one with the verbosity mode off (pentest 0) and one with the verbosity mode on (pentest 17). This assures that both kinds of logging behaviour are captured. It was decided to use the Drain algorithm (He et al., 2017) for this because it has decent performance on Linux Logs and best performance overall of the log key extraction algorithms.

3.4.1 Drain for Log Key Extraction

The Drain algorithm is used to extract log keys from the logs. A log key is a reduced representation of a log-line with variables removed. Formally, let perfect log key extraction in the case of this work be defined as: let L be the language which contains any combination of substrings over every single existing log in the universe, including the empty string ϵ. Given a string $s \in L$ and $k \in \mathbb{N}$, constant substring $c \in L$ and variable substring $v \in L$. Let

$$s = \sum_{i=0}^{k} (c_i \cup v_i)$$

.

Let LKE_p be a perfect log key extractor with replacement string $r =< * >$

$$\forall_{s_1 \in s} \exists_{s_2 \in s \wedge s_2 \neq s_1} \forall_{(c_1, v_1 \in s_1), (c_2, v_2 \in s_2)} (c_1 = c_2 \wedge v_1 \neq v_2 \Rightarrow LKE_p(s_1) = \sum_{i=0}^{k} (c_i \cup r))$$

In practice log key extractors are not perfect and will make mistakes, this can be due to not knowing the full extend of the universe of log-lines and cases where there are multiple variables in a row. Drain has an accuracy of 0.690 on Linux logs (He et al., 2016; Zhu et al., 2019). While both SHISO and LenMa log key extractors perform better at an accuracy of 0.701 on Linux logs, the difference with Drain is minor. Moreover Drain is the best performing log key extractor overall with an average accuracy of 0.865 over any type of log in the logpai benchmark (Zhu et al., 2019), versus respectively 0.669 and 0.721 for SHISO and LenMa. It was decided to use Drain over SHISO and LenMa for two reasons. First, because the STEP logs might not exactly mimic the Linux logs used in the benchmark. Second, because creating a benchmark of different log key extraction methods for the STEP logs is outside the scope of this work. This is due to the amount of time required versus potentially little gain.

3.4.2 Data Preparation Methods from Natural Language Processing

For NLP methods additional data preparation methods are used. This includes limiting the length of log-lines to a fixed amount, tagging log-lines with ⟨BOS⟩ and ⟨EOS⟩ characters, removing stop words, tokenization and removing punctuation. In the case of character based methods each character can be transformed to its ASCII value. The extent to which these data preparation methods are used varies over the different experiments run.

3.4.3 Other Data Preparation Methods

Another data preparation method that is used, entails removing duplicate log-lines. Some logs have a large amount of non-anomalous duplicate log-lines that can have a big influence on possible outcomes of the methods used when trying to find point anomalies. To cope with this it was decided to remove duplicate log-lines. While this does remove those collective anomalies that rely on repetition of log-lines, in the case that we are only searching for point anomalies we would not recognize those as collective anomalies anyway.

3.4.4 Data Preparation Steps

The main data preparation steps are:

1. Parse the syslog which is roughly structured like: "⟨date⟩ ⟨time⟩ ⟨operating system⟩ ⟨process⟩ ⟨additional process information⟩ ⟨message⟩" as shown in Fig 3.2. Then using a regular expression to remove the date, time, operating system and any additional process information. The results are log-lines in the form "⟨process⟩ ⟨message⟩".

2. Perform log key extraction using Drain.

3. Shortening the log-lines to a fixed length.

4. Tagging log-lines with ⟨BOS⟩ and ⟨EOS⟩ characters at the start and end.

An example of the resulting log-lines after these main data preparation steps is given in Fig. 3.3.

```
Jul  4 11:41:44 [redacted] systemd[1]: Starting Time & Date Service...
Jul  4 11:41:44 [redacted] systemd[1]: Listening on Load/Save RF Kill Switch Status /dev/rfkill Watch.
Jul  4 11:41:44 [redacted] systemd[1]: Created slice system-systemd\x2dbacklight.slice.
Jul  4 11:41:44 [redacted] systemd[1]: Starting Load/Save Screen Backlight Brightness of backlight:intel_backlight...
Jul  4 11:41:44 [redacted] systemd[1]: Starting Load/Save Screen Backlight Brightness of backlight:acpi_video0...
Jul  4 11:41:44 [redacted] systemd[1]: Starting Hold until boot process finishes up...
Jul  4 11:41:44 [redacted] systemd[1]: Starting Terminate Plymouth Boot Screen...
Jul  4 11:41:44 [redacted] systemd[1]: Started Login Service.
Jul  4 11:41:44 [redacted] kernel: [   11.877002] Bluetooth: Core ver 2.22
Jul  4 11:41:44 [redacted] kernel: [   11.877017] NET: Registered protocol family 31
Jul  4 11:41:44 [redacted] kernel: [   11.877018] Bluetooth: HCI device and connection manager initialized
Jul  4 11:41:44 [redacted] kernel: [   11.877021] Bluetooth: HCI socket layer initialized
Jul  4 11:41:44 [redacted] kernel: [   11.877024] Bluetooth: L2CAP socket layer initialized
Jul  4 11:41:44 [redacted] kernel: [   11.877029] Bluetooth: SCO socket layer initialized
Jul  4 11:41:44 [redacted] systemd[1]: Started Load/Save Screen Backlight Brightness of backlight:intel_backlight.
Jul  4 11:41:44 [redacted] systemd[1]: Started Load/Save Screen Backlight Brightness of backlight:acpi_video0.
Jul  4 11:41:44 [redacted] dbus[736]: [system] Successfully activated service 'org.freedesktop.timedate1'
```

Fig. 3.2: An example of log-lines from pentest log 0 without the main data preparation steps performed.

```
<BOS> systemd Starting Time & Date Service... <EOS>
<BOS> systemd Listening on Load/Save RF Kill Switch Status /dev/rfkill Watch. <EOS>
<BOS> systemd Created slice <*> <EOS>
<BOS> systemd Starting Load/Save Screen Backlight Brightness of <*> <EOS>
<BOS> systemd Starting Load/Save Screen Backlight Brightness of <*> <EOS>
<BOS> systemd Starting Hold until boot process finishes up... <EOS>
<BOS> systemd Starting <*> Plymouth Boot Screen... <EOS>
<BOS> systemd Started Login Service. <EOS>
<BOS> kernel Bluetooth: Core ver <*>.<*> <EOS>
<BOS> kernel NET: Registered protocol family <*> <EOS>
<BOS> kernel Bluetooth: HCI device and connection manager initialized <EOS>
<BOS> kernel Bluetooth: <*> socket layer initialized <EOS>
<BOS> kernel Bluetooth: <*> socket layer initialized <EOS>
<BOS> kernel Bluetooth: <*> socket layer initialized <EOS>
<BOS> systemd Started Load/Save Screen Backlight Brightness of <*> <EOS>
<BOS> systemd Started Load/Save Screen Backlight Brightness of <*> <EOS>
<BOS> dbus [system] Successfully activated service <*> <EOS>
```

Fig. 3.3: An example of log-lines from pentest log 0 with the main data preparation steps performed.

The optional data preparation steps in no particular order are:

- Removing duplicate log-lines.
- Removing stop words and punctuation.
- Tokenizing words and transformation to vocabulary values.
- Transforming characters to ASCII values.

3.5 Modeling Development and Selection

Two modeling techniques are selected, in this work they are referred to as deep anomalous collective events prediction and deep anomalous log-line prediction.

3.5.1 Deep Anomalous Collective Events Prediction

The first method that is selected is deep anomalous collective events prediction which is an implementation of the fourth experiment outlined in the master thesis "Anomaly Detection for Application Log Data." by Aarish Grover (Grover (2018)). The goal of deep anomalous collective events prediction is to use an auto encoder deep learning model to find multisets of events that constitute collective anomalies. The model uses two layers of LSTMs which are interfaced by an LSTM-based Auto Encoder. The two-layer LSTM is originally used by Grover in his third experiment; the fourth experiment adds the LSTM-based Auto Encoder as an interface to this two-layer LSTM. The author achieved their best performance with the model of the fourth experiment finding anomalies in HDFS (Hadoop Distributed File System) (Xu et al., 2009) and BGL (Blue Gene/L, logs from a super computer) (Oliner and Stearley, 2007) labeled datasets. Their model of the fourth experiment achieves an F1 score of 0.88 and 0.89 for the HDFS and BGL datasets respectively. The model of the third experiment containing only the two-layer LSTM performs slightly worse with an F1 score of 0.84 and 0.87 for the HDFS and BGL datasets respectively.

Deep Anomalous Collective Events Prediction Method

Deep anomalous collective events prediction works by classifying multisets of events as either normal or anomalous. Every unique log-line is an event e from the universe of all events E. In practice only events that appear in any of the logs used are considered, i.e. $E_{log} \subseteq E$. The multisets of events are generated by combining a number of events based on some grouping criterion. In the case of HDFS logs for example this grouping criterion is one multiset of events per HDFS block, another used criterion could be a specific time frame. The multisets of events are converted to an event count vector which is a vector over every $e \in E_{log}$ counting every occurrence of each event in the multiset. These event count vectors are used as input for the fourth experiment model outlined by Grover which consists of two layers of LSTMs which are interfaced by an LSTM-based auto encoder, the structure of the model is shown in Fig. 3.4. The goal of the model is for the auto encoder to learn a representation of the typical shape of the event count vectors of normal blocks, such that when an anomalous event count vector is attempted to be reproduced by the auto encoder it incurs a high reconstruction error. This reconstruction error can then be used as a metric for finding the most anomalous event sequences. Deep anomalous collective events prediction is useful for finding collective anomalies rather than point anomalies. This is because only collections of log-lines are being classified as either normal or anomalous rather than individual log-lines.

Data Preparation Steps for Deep Anomalous Collective Events Prediction

It is important to perform log key extraction on the logs for deep anomalous collective events prediction. This is because the size of E_{log} must be kept as small as possible in order to not explode the size of the event count vector making prediction infeasible with this method. As explained in Section 3.4 the Drain algorithm is used for log key extraction in this work. Data preparation steps from NLP, like stop words removal, are not as important for deep anomalous collective events prediction because log-lines are converted to unique event ids rather than the text being used explicitly.

Deep Anomalous Collective Events Prediction Limitations

There are several issues with this method that make it infeasible to solve the business problem. First, the model can only predict for the events that are part of E_{log}. If some new log comes in for

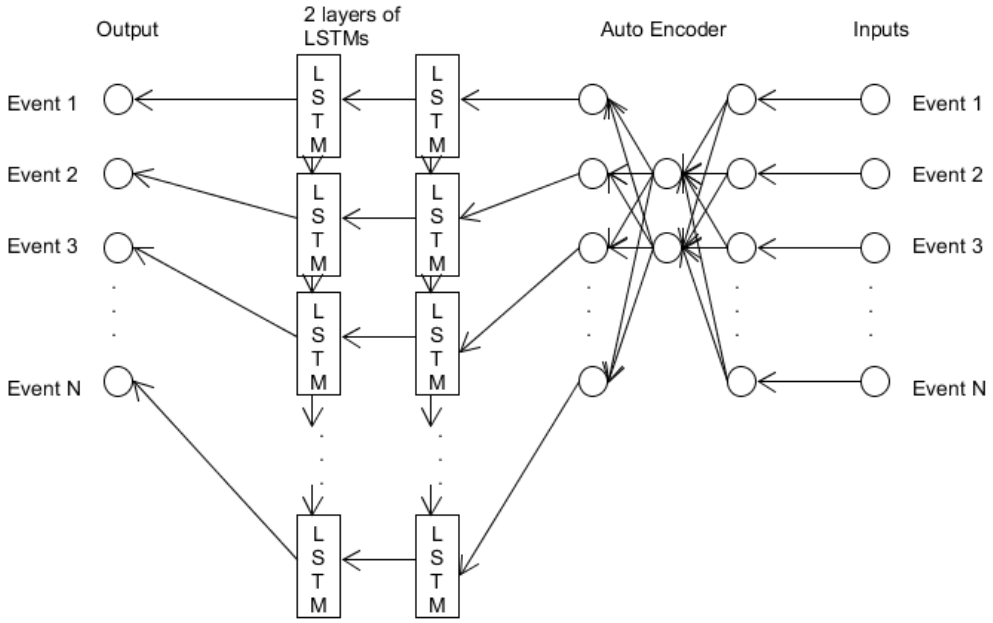


Fig. 3.4: Deep anomalous collective events prediction LSTM Autoencoder model.

classification that contains events which are not in E_{log}, then the whole model has to be retrained to capture the change in input dimensions. While it is possible to drop events that do not exist in the model from the input, these events are bound to be interesting because they are more likely to depict anomalous events, due to not existing in normal logs. Second, while Drain assures that the same log keys should always be hashed to the same event, because Drain does not have a 100% accuracy, it is possible for log keys that should hash to the same event id to hash to different event ids. If there are many incorrect hashes it will hurt the performance of the model as well as exploding the size of E_{log}, creating potential memory problems. Drain also has a relatively low performance for Linux logs overall due to their complexity, so the amount of incorrect hashes will be noticeable. Third, STEP logs do not have a strong grouping criterion. Grouping based on fixed block sizes will just group parts together that should not go together. Grouping based on time might be possible but many events happen at time 0 because the timer has not been activated yet. With many events interleaving due to parallel running processes, this might group events together that should not be together. The strongest grouping is probably a grouping based on process, possibly in combination with time. Finally, by thoroughly studying the structure of the syslogs it was concluded that syslogs might be too complicated to really benefit from this method, a lot of events will not be distributed in the same way because different processes run parallel and can be run and generate logs at the same time. This method is a better fit for logs which are only monitoring a single process in which all events that can happen are known. The number of events is just too large for this method to work well with STEP data. Drain mistakes will decrease the performance and there are no strong event grouping available. This method can possibly work if there is a specific process one is interested in monitoring, and this process has some additional identifier which can be grouped together e.g. blocks created or user id. Because of these limitations it was decided not to invest any more time into implementing this method.

Comparison with HDFS logs

For HDFS logs this method works because the number of events is very limited at only about 30 events total, versus hundreds if not thousands for STEP logs. Because there are so few events Drain will perform much better, Drain has an accuracy of 0.998 for the HDFS dataset in the logpai benchmark versus 0.701 on the Linux dataset (Zhu et al., 2019). Finally, HDFS log-lines are easily grouped on blocks and blocks are labeled as either normal or anomalous. Therefore collective anomaly detection for all events in a block makes sense, as this is exactly what anomaly detection consists of in HDFS datasets.

3.5.2 Deep Anomalous Log-Line Prediction

The second model that was selected is deep anomalous log-line prediction. Because of the limitations that the deep anomalous collective events prediction model has it was decided to look at another type of model, the deep anomalous log-line model that was selected is fit to find point anomalies but has no means to find collective anomalies or contextual anomalies. Several language models for finding anomalies in system logs are outlined in [Brown et al. (2018)]. The simplest model is the Event Model (EM). EM uses an unidirectional LSTM to predict the probability of each next token in a sequence of tokens from a tokenized sentence. Tokens can be either words or characters. The LSTM assures that long-term dependencies on tokens from the past are remembered by the model. For every model the negative log-likelihood is used as a loss function. EM is improved by replacing the unidirectional LSTM by a bidirectional LSTM, this model is called the Bidirectional Event Model (BEM). In a bidirectional LSTM there is an additional LSTM that predicts the tokens from the ending to the beginning in reverse order. The results from both sides are combined to remember long-term dependencies on tokens from both the past and the future for the prediction of every token. The BEM outperforms the EM for both word and character-based models on the LANL (Kent, 2016)(Los Alamos National Laboratory) dataset benchmark performed in [Brown et al. (2018)]. To further improve the performance of the EM and the BEM, the authors propose to create a tiered model that adds a higher layer contextual LSTM which chains log-lines together based on some grouping criterion e.g. every log-line from a specific user. The contextual LSTM predicts on log-line level rather than token level and takes in the result of the last log-line as well as a summary of the results of the lower level EM and BEM. This allows for finding contextual anomalies by taking into account the past log-lines. The tiered models are called the Tiered-Event Model (T-EM) and the Tiered-Bidirectional Event Model (T-BEM). The main contribution of [Brown et al. (2018)] is the improvement of the T-EM and the T-BEM by adding an attention layer. The output of the LSTM is summarized by the attention layer rather than the mean. The tiered models with attention are called the Tiered Attention-Event Model (TA-EM) and the Tiered Attention-Bidirectional Event Model (TA-BEM). While not defined as such in [Brown et al. (2018)], in this work the event models with attention but without the tiered architecture are called the Attention-Event Model (A-EM) and the Attention-Bidirectional Event Model (A-BEM). The authors of [Brown et al. (2018)] tested the A-EM with four different implementations of attention, with most attention implementations the A-EM slightly outperforms the EM on word and character tokens, while the TA-EM and the TA-BEM barely see an improvement versus the T-EM and the T-BEM on word and character tokens. The best performing models on the LANL dataset are the TA-BEM for word tokenization with an AUC of 0.988 and the T-BEM with character tokenization with an AUC of 0.992.

The Advantages of Attention Mechanisms

The main reason to use an attention layer is to provide a means of interpretability for the LSTM models outlined in the previous section. The attention weights can be used to provide information on relational mapping between features. In combination with feature importance based on a percentage chance for a token during prediction, this provides a means of interpretability for anomalies found by LSTM models in system logs.

Choice for Deep Anomalous Log-Line Prediction Models Used

It was decided to not use the tiered models due to the complexity of implementation in combination with the time limit. So it was decided to use the other models outlined in Section 3.5.2 the EM and the BEM as well as the attention models: A-EM and A-BEM. Models using both character and word tokens are used. An example of the BEM architecture used is shown in Fig. 3.5. In comparison the EM architecture only uses the first column of LSTMs from the inputs. The A-EM and the A-BEM have a column of attention units where the output of the LSTM(s) goes through before reaching the output node. As log-line anomaly score the mean over the probabilities for individual tokens will be used. This is a measure of how common a certain log-line is within the model. It is assumed that log-lines with a low anomaly score are not (well) known by the model and are therefore more anomalous. The advantage of the word-based model over the character-based model is that it is easier to use SHAP values for words than for characters.

Character Token Anomaly Definition

In the case of character tokens let A be the ASCII alphabet and let the language length N be defined as: $N = 120$. Let the language L_N be the set of all permutations of strings of N characters that can be created by sampling with replacement from A. Let g be a function mapping some $x \in A^n$ representing the set of all permutations of strings up to N characters to a single ASCII character $y \in A$. Function g approximates the probability of x and y happening simultaneously and is defined as: $g : A^n \to A$, $g(x) \approx P(x, y)$. The goal is to assign a log-line $l \in L_N$ an anomaly score in the range $[0, 1]$. Training examples are defined as: $T = \{(x_1, y_1), ..., (x_N, y_N)\}$. Given character $c_i \in l$ where i is the index of c_i in l, let $x_i = \sum_{i=1}^{L-1} c_i$ and $y_i = c_{i+1}$. For the reverse case of the bidirectional LSTM let $x_i = \sum_{i=1}^{L-1} c_{L+1-i}$ and $y_i = c_{L-i}$. The anomaly score δ_i^{char} of a single character at position i is defined as:

$$\delta_i^{char} = g(x_i)$$

Note that high values for $g(x_i)$ indicate that a character is normal and belongs to the learnt model, low values indicate that a character is abnormal and do not seem to be generated by the learnt model. The name "anomaly score" is therefore slightly misleading in this case because one would expect high scores to be anomalous. "normality score" would fit it better. An anomaly score where anomalous instances are ranked closer to 1 and normal instances are ranked closed to 0 can be easily achieved by computing the anomaly scores as $1 - g(x_i)$. However, in our case we use the term anomaly score to refer to $g(x_i)$ and not $1 - g(x_i)$ with lower scores indicating anomalies and high scores indicating normal instances.

Word Token Anomaly Definition

In the case of word tokens let V be the vocabulary. Let the language L_V be the infinite set of all permutations of words from V sampled with replacement. Let g be a function mapping some

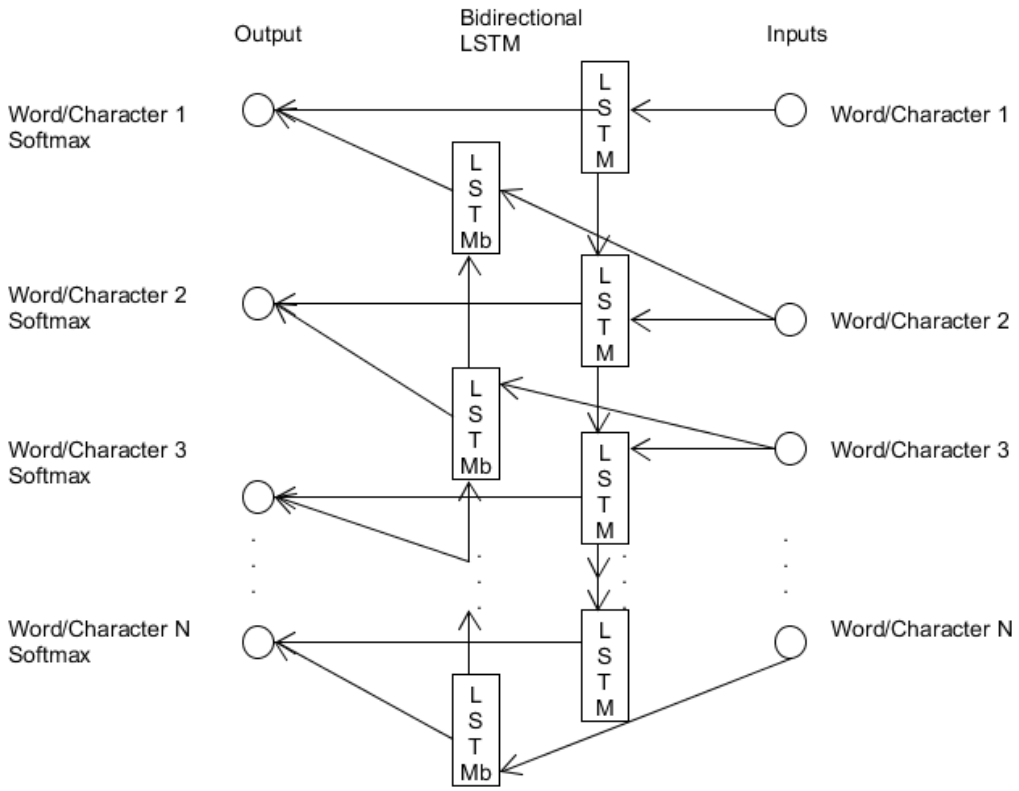


Fig. 3.5: Deep anomalous log-line prediction model (BEM).

$x \in V^n$ representing the set of all permutations of strings up to N words to a single word $y \in V$. Function g approximates the probability of x and y happening simultaneously and is defined as: $g : V^n \to V$, $g(x) \approx P(x, y)$. The goal is to assign a log-line $l \in L_V$ an anomaly score in the range $[0, 1]$. Training examples are defined as: $T = \{(x_1, y_1), ..., (x_N, y_N)\}$ where N is the amount of words in log-line l, given word $w_i \in l$ where i is the index of w_i in l, let $x_i = \sum_{i=1}^{L-1} w_i$ and $y_i = w_{i+1}$. For the reverse case of the bidirectional LSTM let $x_i = \sum_{i=1}^{L-1} w_{L+1-i}$ and $y_i = w_{L-i}$. The anomaly score δ_i^{word} of a single word at position i is defined as:

$$\delta_i^{word} = g(x_i)$$

Log-Line Anomaly Definition

Given the previously defined expressions and the anomaly score for token i in log-line l, δ_i^q for $q \in \{char, word\}$. The log-line anomaly function $\delta_l^{log-line,q}$ is defined as

$$\delta_l^{log-line,q} = \frac{1}{(N-1)} \cdot (\sum_{i=1}^{N-1} \delta_i^q)$$

Log Anomaly Definition

Given the previously defined expressions and log-line l_i where i is the index of the log-line within the log and LL is the amount of log-lines in the log. The log anomaly function $\delta^{log,q}$ is defined as

$$\delta^{log,q} = \frac{1}{LL} \cdot (\sum_{i=1}^{LL} \delta_{l_i}^{log-line,q})$$

Data Preparation Steps for Deep Anomalous Log-Line Prediction

The Drain algorithm can be used to simplify the vocabulary used for word tokens. Data preparation steps from NLP are important because it simplifies the vocabulary and removes words and punctuation that appear often but are not very useful for prediction.

3.5.3 Limitations

Models using the tiered architecture would probably outperform the best models without tiered architecture. Since it was decided to not use the tiered architecture in this work there is likely to be a performance increase if tiered models were used in future work. The negative log-likelihood would likely yield a better anomaly score than the mean over the probabilities for individual tokens. This was realized late in the realization of this work, therefore it cannot be easily changed and will be kept for future work.

3.5.4 Other Anomaly Prediction Methods

Three other anomaly prediction methods will be used, clustering using the Normalized Compression Distance (NCD), Inter-Log Anomaly Detection using Term Frequency-Inverse Document Frequency (TF-IDF) and naive substring filtering.

Clustering with Normalized Compression Distance

Normalized Compression Distance (Cilibrasi and Vitanyi, 2005) is a similarity measure between two different data objects, which compares compressions of data objects individually and concatenated. NCD is defined by:

$$NCD(x,y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}}$$

Where C is a compressor and $C(x)$ is the binary length of the compression of x, $C(y)$ is the binary length of the compression of y. $C(xy)$ is the binary length of the compression of the concatenation of x and y. In this work Bzip2 is used for compression because it is the best performing. Gzip can be used for a significant speed-up but slightly worse performance.

Inter-Log Anomaly Detection using TF-IDF

The "Term Frequency-Inverse Document Frequency" (TF-IDF) metric can also be used to find inter-log anomalies. TF-IDF is a metric of how concentrated into relatively few documents the occurrences of a given word are (Rajaraman and Ullman, 2011). It is the multiplication of the TF_{ij} metric with the IDF_i for a given word i in document j. The term frequency is the normalized frequency f_{ij} of a word i in a specific document j and is defined as $TF_{ij} = \frac{f_{ij}}{max_k f_{kj}}$ due to the normalization the word with the highest frequency in the document has a term frequency of 1. The term frequency is therefore a measure of how frequent a specific word is compared with the other words in a document. The inverse document frequency is a measure of how uncommon a word i is amongst the complete corpus of N documents. Suppose n_i is the number of documents which contain the word i within the complete corpus of N document, then the inverse document frequency is defined as $IDF_i = log_2(N/n_i)$. Uncommon words have high IDF scores while common words have low IDF scores. The TF-IDF score is given by: $TF_{ij} * IDF_i$. The words with the highest TF-IDF scores are often those that uniquely characterize a specific document. In this work TF-IDF is not used for its intended purpose of finding words that uniquely characterize a specific document, but rather as an additional anomaly measure. The anomaly score of a document is defined as the TF-IDF score of the "$\langle EOS \rangle$" End of Sequence character, note that this score is identical to the "$\langle BOS \rangle$" Begin of Sequence character TF-IDF score. TF-IDF weighting of the vector space representation helps make the pairwise comparison between vectors more meaningful, as it informs about the amount of weight a dimension deserves globally (inter-log) and locally (intra-log). It is computationally expensive to carry out all pairwise comparisons between vectors though, but clustering for anomaly detection using TF-IDF is possible. However, during experimentation with TF-IDF it was noticed that just the TF-IDF score of the "$\langle EOS \rangle$" End of Sequence character already provided a very decently performing anomaly measure at a very low computational cost. Hence it was decided to use this TF-IDF score as an additional dimension for the models. This is not the intended usage of TF-IDF and it is strange that it would perform so well.

To find out why it performs decently well as an anomaly detection measure it is important to look at the details of how the TF-IDF is computed in this case. TfidfTransformer from Scikit-learn is used. [1] It computes the IDF score as: $IDFTransformer_i = log_2(1 + N/1 + n_i) + 1$ which is slightly different from our definition as it tries to prevent division by zero. Note that the "$\langle EOS \rangle$" End of Sequence character has a couple of key properties in this case:

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

1. "$\langle EOS \rangle$" appears exactly once in every log-line.

2. Because of 1): "$\langle EOS \rangle$" appears in every log.

3. Because of 2): The n_i of "$\langle EOS \rangle$" is N.

Now taking these properties into account it is found that $IDF_i = log_2(N/N) = 0$, which means TF-IDF is always equal to 0 in the original definition. However, in the case of the Scikit-learn definition of IDF it is found that $IDFTransformer_i = log_2(1 + N/1 + N) + 1 = 1$ hence we find that the TF-IDf score is always equal to the term frequency TF, and the IDF score is irrelevant. Since the TF is always bounded between 0 and 1, it always returns a score that is in the same dimensions as the other anomaly scores defined from Section 3.5.2 onwards. Because the TF is defined as the frequency percentage of a word compared with the highest frequency word in a log, this also means that there is some word that, on average, appears more than once per log-line for every log as otherwise "$\langle EOS \rangle$" would have a TF score of 1, which hasn't happened for any of the logs. Because it appears that this metric has some worth in anomaly detection, what it says is that a log is more anomalous based on how often the most frequent word appears compared with the amount of log-lines. If the word appears much more often per log-line it is more anomalous and if it appears less often per log-line it is less anomalous. Since this is kind of a hacked metric the worth of it outside these specific circumstances is probably minimal. As for the reason why this metric gives good performance, perhaps there is some relationship between a particular word appearing more often and the log being anomalous. It's also possible that the anomalous logs simply have some word being more common because the logs were generated at different times when this word would appear more commonly throughout the logs. It seems like there might be a correlation between the size of the log and whether they belong to one of the anomalous datasets. For example all pentest logs that use the verbose way of logging also have a larger size than most if not all of the normal logs. It might be the case that the TF-IDF score of the "$\langle EOS \rangle$" character simply measures the size of the log and can thus be replaced with the cheaper option of simply using the (normalized) size of a log as a metric. There is no answer for this at the moment, more research would be required.

Naive Substring Filtering

A simpler albeit naive way of finding anomalies is to automatically mark any log-line that contains a word from some preselected list of words. While this is a simpler way of finding anomalies it is very easy to a get a lot of false positives if one does not look at the context. Naive substring filtering can be used as a baseline of model performance. In practice this list of words was created: ["error", "invalid", "bug", "corrupt", "failed", "failure"]. If a log-line contains any word in this list it will be automatically marked as an anomaly. Another approach can be to give an anomaly score penalty to log-lines containing any of these words, in this way it is not as strict as marking lines anomalous without context.

3.6 Modeling Interpretability for Bidirectional LSTM

There is need for some way to understand the decisions the bidirectional LSTM makes. It can be useful for experts to know why the LSTM generates a certain prediction. However since softmax output is used finding a good interpretable anomaly alert processing framework is challenging.

3.6.1 Interpretable Anomaly Alert Processing for Bidirectional LSTM

Interpretable anomaly alert processing methods are investigated to use for the deep anomalous log-line prediction models. Interpretable anomaly alert processing for neural networks is a relatively new research domain and there is not a lot of research. Because neural networks are basically a black box, interpretability is tricky. There are some methods for convolutional neural networks that work reasonably well like saliency maps. This is most likely because convolutional neural networks often uses images which are easy to understand so saliency maps also make sense to humans. Two methods of interpretable anomaly alert processing for LSTMs were investigated, attention and Seq2Seq-Vis (Strobelt et al., 2018).

Attention

By adding an attention layer to the bidirectional LSTM, the attention layer can be used to visualize the dependencies on which the score of a word depends the most. In the end, the attention layer decreased the performance of the model and increased the training time of the model significantly. Because it was not possible to use the attention layer, attention based interpretability could not be implemented.

SHAP Values

SHAP (SHapley Additive exPlanations) (Lundberg and Lee, 2017) is a method to find explanations for individual predictions in a machine learning algorithm. It uses Shapley values from game theory. SHAP calculates the individual contributions of each feature for a prediction. This allows to explain which features were most important for a model to come to its prediction. SHAP cannot be used to provide explanations for the model because the model provides higher dimensional data as output while DeepExplainer expects vector data.

Seq2Seq-Vis

Seq2Seq-Vis is a visualization tool for sequence to sequence models. Because a full sequence to sequence model is not being used Seq2Seq-Vis could not be used. It was also decided that the interpretability options are too complicated and not very understandable when used with system logs for lay persons.

Conclusions Interpretable Anomaly Alert Processing for Bidirectional LSTM

Our study of the applicability of the existing approaches showed that neither of them is applicable in our case, directly or with small adaptations. The reasons for that are the following. The attention layer decreased the performance of the model, SHAP DeepExplainer could not be used because the model in its current form is incompatible and Seq2Seq-Vis could not be used because a full sequence to sequence model is not being used. Instead of this and in the interest of time, it was decided to implement an ad hoc visualization which turned out to be useful for the STEP technical expert and tester at Carapax IT. The ad hoc visualization uses the anomaly scores directly to colour words and log-lines according to their abnormality or normality.

3.7 Evaluation

In this section the evaluation metrics which are used are outlined.

3.7.1 Evaluation for Anomaly Detection

We decided to use the AUC score as the main evaluation metric. However, due to an error in computing the AUC scores, these scores were not reliable and had to be removed. Since a lot of these scores were made with older versions of the model, it would be a challenging and time-consuming task to find the correct versions of the model in which these scores were originally generated. Therefore it was decided to report the balanced accuracy and F1 score instead as these scores were calculated correctly. The balanced accuracy is a decent metric for imbalanced datasets and the F1 score combines the precision and recall into a single score. The AUC or pAUC are a better metric, but due to the error in computation it was decided to drop them from this work and as such they are kept for future work.

Flagging criteria

In order to evaluate the performance of the method, it needs to be decided what the criteria are to flag log-lines and logs as anomalous. There are multiple possibilities for flagging anomalies.

- Flag point anomalies only without session context.
- Flag collective or contextual anomalies as a sequence of log-lines.
- Flag logs as point anomalies then return point/collective/contextual anomalies for those sessions only.

Given that in an anomaly detection problem it is inevitable that one will find many false positives, it was decided that the session level would be looked at first. If sessions would be disregarded and just point anomalies/contextual anomalies would be flagged on every session then even the sessions that are not anomalous would contain log-lines that would be erroneously flagged as anomalies. If one first looks at the most anomalous sessions and only of those return the flagged anomalies, it reduces the workload for experts using the system. The downsides are that it introduces an attack vector for fraudsters.

Finding Optimal Error Threshold

To find the optimal error threshold for the intra-log anomaly detection the performance metrics m are measured at every threshold step t from 0 to 1. This gives a metric over error threshold graph. The optimal error threshold is defined as $m_{opt} = max(m_t)$. Different performance metrics can have different optimal error threshold values. In practice, however, there is no way to calculate the optimal error threshold, hence a different method of determining the error threshold is required. Either manually based on which values worked in other cases, assuming there is an universal optimal error threshold, or by setting it automatically based on some metric.

3.7.2 Evaluation for Alert Processing

A small study will be performed with the technical expert and the tester from Carapax IT in order to determine if the interpretability solution is useful for the users of the model. This study gathers ad hoc evidence from the technical expert and the tester from Carapax IT to determine whether the ad hoc visualization is useful for them. Because the visualization is primarily meant only for the technical expert and the tester from Carapax IT, numerical evidence for the worth of the visualization is not as important as it would have been, had the visualization been written for a larger group of people. A larger study that is meant to provide numerical evidence is left to future work.

Evaluation for Alert Processing Future Work

A larger study that is left for future work, because of time constraints, is detailed in this section. The goal of this study is to achieve a better evaluation of the ad hoc visualization which is supported by numerical evidence, rather than just the ad hoc evidence provided by the small study. The research questions are: "Does using the ad hoc model visualization allow the experts to check the logs faster than without using the visualization" and "Does using the ad hoc model visualization allow the experts to find more anomalies in the logs".

1. Select 4 logs: 2 pentest logs, 1 fault log and 1 hackathon log.
2. Randomly select 2 logs from this set with ad hoc model visualization and 2 without.
3. Ask the technical expert and the tester from Carapax IT to check these 4 logs for anomalies.
4. Notate the log-lines chosen as anomalous and notate the time taken to check each of the logs.
5. Check and notate how many of the log-lines were correctly identified by the technical expert and the tester from Carapax IT.
6. After 4 weeks, repeat step 2 through 5 but reverse the logs that are with the ad hoc model visualization and those without.
7. Verify whether logs that were checked using the ad hoc model visualization have faster checking times and a larger amount of correct anomalies found than the logs that were checked without using the ad hoc model visualization.

The reason to let the technical expert and the tester from Carapax IT check two logs with ad hoc model visualization, and two without, is to reduce bias from starting with one or the other. Since the technical expert and the tester from Carapax IT will be checking the same logs twice—because otherwise it is not possible to see if there are improvements—there needs to be a sufficient time interval between checking the same log. This is to avoid them remembering where the anomalies were in the logs. Because two of the logs were checked in random order with the ad hoc visualization and two without, and after 4 weeks the same is done but reversed again in random order, we hope this will eliminate the possible bias in the study.

3.8 Deployment

The deployment of the model is not a main focus of this work, a potential implementation of the final model into STEP in compliance with the business requirements of the client as outlined in Section 1.2.5 is given in Section 4.1.6.

Chapter 4

Experiments and Results

The first goal of the experimental study is to investigate whether or not the modeling techniques outlined in Section 3.5 can be used to reliably find logs with hack attacks and logs with faults amongst a large amount of normal logs without having a lot of false positives. This includes investigating the best decision boundary for the anomaly scores returned by the modeling techniques. The second goal is to create and evaluate an ad hoc visualization of the results from the best method identified. The ad hoc visualization is evaluated with a small ad hoc study with the technical expert and the tester from Carapax IT. Finally the first goal is revisited and the chosen model is further optimized based on the feedback from the technical expert and the tester from Carapax IT given during the small ad hoc study.

4.1 Anomaly Detection

A number of different experiments are run, mostly different versions of the deep anomalous log-line algorithms were compared. The EM is compared with the BEM, character tokens are compared with word tokens and the impact of attention on the model is examined.

4.1.1 Preliminary Experiments with Pentest Logs for Character-Based Methods

For the first experiments only the TU/e dataset and the pentest dataset were available. An overview of the first experiments using only the TU/e dataset, pentest dataset, and later the partial fault dataset, is given in Appendix B. We discovered that the character-based EM with duplicate log-lines, using naive substring filtering and employing intra-log anomaly detection, can achieve a balanced accuracy of 0.8419 on pentest 0 and 0.7671 on pentest 17. The character-based EM with duplicate log-lines removed, naive substring filtering and employed for inter-log anomaly detection, shows that there is indeed a subset of pentest logs that can be easily distinguished from TU/e logs by their $\delta^{log,char}$ score. But there is also a subset of pentest logs that is indistinguishable from TU/e logs using this model. We also tried to add an attention layer to the character-based EM, however the performance of this model with attention layer was much lower than the performance of this model without attention layer. Next, the EM was replaced with the BEM and naive substring filtering was disabled as there are often log-lines that include those substrings that are not anomalous. An isolation forest model was trained to serve as a baseline comparison to our models. It was found that the BEM could outperform the isolation forest in balanced accuracy and F1 score metrics at specific $\delta_l^{log-line,char}$ threshold

intervals. Furthermore, we decided to keep using the BEM rather than the EM because of the historical evidence from the original paper that shows that BEMs are always better than EMs, even though our own study was not as conclusive. The isolation forest can distinguish a subset of the pentest logs from TU/e logs but is unable to distinguish the logs of the partial fault dataset from the TU/e logs. The character-based BEM however can distinguish both a subset of pentest logs as well as all logs from the partial fault dataset with only a small amount of false positives. Finally we trained a support vector machine in order to investigate if we can automatically generate a good decision boundary, because the $\delta^{log,char}$ scores differ wildly based on how long the model has been trained for and do not have an obvious default decision boundary value like the isolation forest has.

4.1.2 Word-based Methods and the Inclusion of the Fault Dataset

Towards the end of our preliminary experiments, the partial fault dataset became available, it is important to note that many of the logs in the fault dataset are quite old. The other 28 logs as mentioned in Section 3.3.1 were added at a later date and are more modern logs. We moved away from character-based methods because they are often expensive to run, considering that every character—up to some character limit per log-line—gets vectorized. We replaced the character-based methods by word-based methods which vectorize every word to some vocabulary, which is much more efficient but—as seen at the end of the intro to Section 3.5.2—can also have slightly weaker performance compared to character-based methods.

Word-based BEM

We replaced the character-based BEM with the word-based BEM and recomputed the inter-log anomaly detection metrics on the word-based BEM. The best results for pentest 0 and 17 for the word-based BEM are shown in Table 4.1—note that the isolation forest baseline still uses the character-based encoding rather than the word-based encoding—with an error threshold of 0.955 and 0.959 respectively. The balanced accuracy scores and F1 scores for both pentest 0 as pentest 17 are higher for the word-based BEM compared with the character-based EM given in Table B.1.

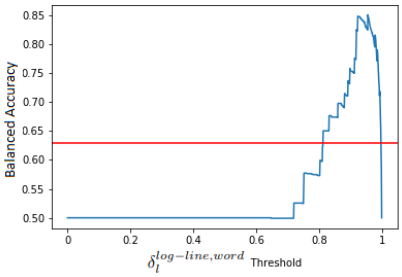
Another word-based BEM (Embedding to 16 dimensions, bidirectional LSTM with 64 cells and a softmax output over the vocabulary size of 10000) was trained on 100 TU/e syslogs for 50 epochs per syslog for a total of 5000 epochs. Then $\delta_l^{log-line,word}$ threshold plots were generated for dataset 0 and 17 as shown in Fig. 4.1.

	pentest 0	pentest 17
Balanced Accuracy	0.8512	0.7980
F1 Score	0.2712	0.5160
Confusion Matrix	[[511 83] [3 16]]	[[618 354] [8 193]]

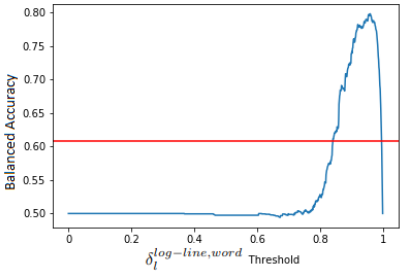
Table 4.1: Performance metrics for intra-log anomaly detection for the word-based BEM on pentest 0 and 17.

Word-based BEM for Fault Logs

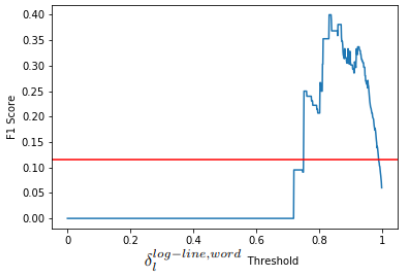
At this point we expanded the partial fault dataset with 19 more recent fault logs, thus creating the full fault dataset. The $\delta_l^{log-line,word}$ scores for these faults logs are plotted together with the pentest and the TU/e logs in Fig. 4.2. All of the logs have their duplicate log-lines removed.



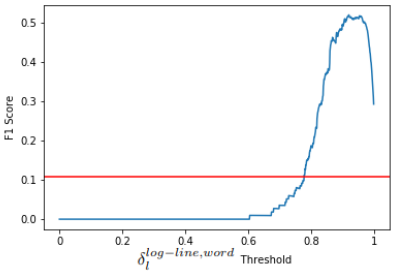
(a) Pentest 0 Balanced Accuracy



(b) Pentest 17 Balanced Accuracy



(c) Pentest 0 F1 Score



(d) Pentest 17 F1 Score

Fig. 4.1: Different $\delta_l^{log-line,word}$ threshold value of pentest 0 and 17 for the word-based BEM. The red line denotes the baseline from the isolation forest model with an anomaly threshold of 0. It can be seen that the word-based BEM outperforms the character-based BEM from Fig. B.3 on all metrics on its optimal $\delta_l^{log-line,word}$ threshold value.

It can be seen that the pentests can still be distinguished from the TU/e logs. The fault logs are closer in average to the TU/e logs but all of them can still be flagged as anomalies if we are okay with a couple of false positive TU/e logs. We trained the support vector machine on these $\delta_l^{log,word}$ scores with the false negative "class_weights" set as "12" instead of "50" and the results can be seen in Fig. 4.3. The performance metrics for this decision boundary are given in Table 4.2. We see that we have a balanced accuracy of 0.8158, ~65% of the 46 anomalous logs are correctly returned as anomalies and only ~2% of the 633 TU/e logs were returned as false positives.

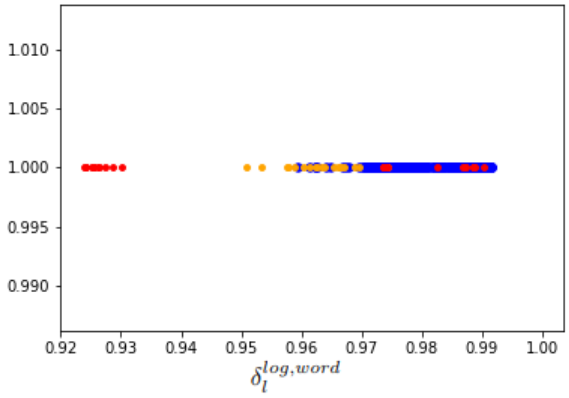


Fig. 4.2: The $\delta_l^{log-line,word}$ scores for the TU/e dataset (blue) versus the new fault dataset (orange) and the pentests (red) for the word-based BEM using only unique events. Most fault logs have low scores which makes them distinguishable at the cost of some false positives.

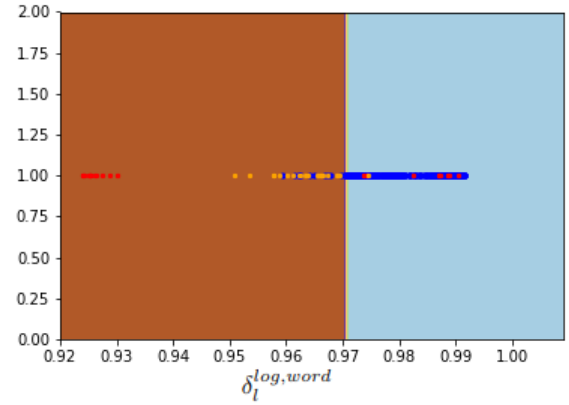


Fig. 4.3: The decision boundary given by the support vector machine on the data of Fig. 4.2. It neatly divides most of the fault logs.

Balanced Accuracy	0.8158
F1 Score	0.6742
Confusion Matrix	[[620 13] [16 30]]

Table 4.2: Performance metrics for inter-log anomaly detection for Fig. 4.3

4.1.3 Clustering with Normalized Compression Distance

Clustering Logs for Inter-Log Anomaly Detection

250 TU/e syslogs and the 24 pentest syslogs are clustered using the normalized compression distance (NCD). A similarity matrix with the NCD between each pair of 274 logs was generated. Spectral clustering [1] was used with 8 clusters, running 100 iterations, using the "eigen solver" "arpack" and "gamma" set to "1" on the similarity matrix. The results are then scaled using multidimensional scaling [2] to 3D and 2D as shown in Fig. 4.4. The dots denote normal TU/e logs and the stars denote anomalous pentest logs. Each colour denotes a different cluster.

We decided to take the red and blue clusters as anomalous, since they contain the most pentest logs. The performance metrics for inter-log anomaly detection are computed as shown in Table 4.3. While this clustering can distinguish 21 out of 24 pentests, almost half of the TU/e logs are returned as false positives as well. With a balanced accuracy of only 0.7115 it does not seem that clustering with the NCD is a very good method for inter-log anomaly detection.

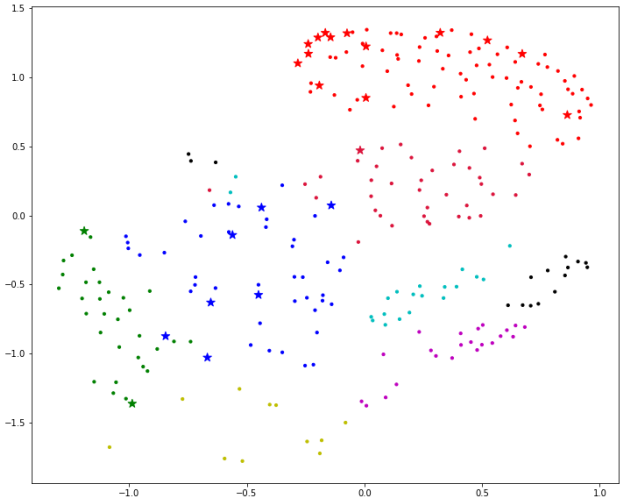
Balanced Accuracy	0.7115
F1 Score	0.2658
Confusion Matrix	[[137 113] [3 21]]

Table 4.3: Performance metrics for inter-log anomaly detection based on clustering with the NCD.

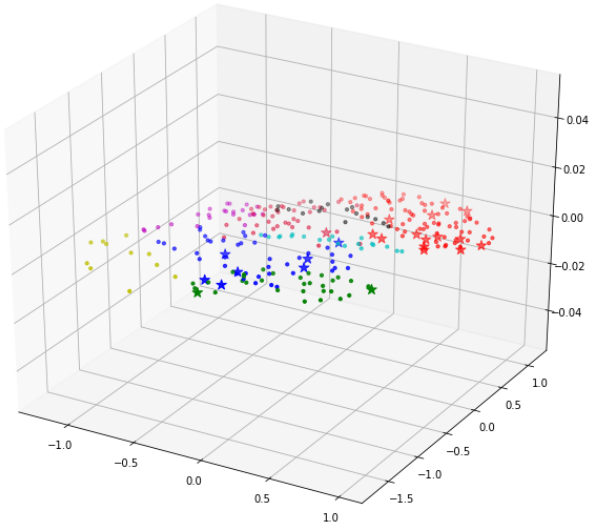
Clustering Log-Lines for Intra-Log Anomaly Detection

The individual log-lines of pentest syslog 0 and 17 are clustered using the NCD. A more clear masked view is also provided with all log-lines that are marked anomalous using the word-based BEM with an optimal balanced accuracy $\delta_l^{log-line,word}$ threshold of 0.955 and 0.959 respectively. The results are shown in Fig 4.5. The two graphs (a) and (b) show a clustering for all log-lines within a syslog while the two graphs (c) and (d) show the masked view which only shows the true anomalous log-lines with a star and the remaining false positive log-lines with a dot. We see that for both pentest syslog 0 and pentest syslog 17 most of the anomalous log-lines are part of the larger red group. It does not seem possible to create a clustering using the NCD that is useful for intra-log anomaly detection on pentest syslog 0 and pentest syslog 17. Since both of our experiments with clustering using the normalized compression distance for inter-log anomaly detection and intra-log anomaly detection were unsuccessful, we conclude that using the normalized compression distance for anomaly detection on whole syslogs and syslog log-lines is not a promising method for anomaly detection for and within Linux syslogs.

[1] https://scikit-learn.org/stable/modules/clustering.html#spectral-clustering
[2] https://scikit-learn.org/stable/modules/manifold.html#multidimensional-scaling

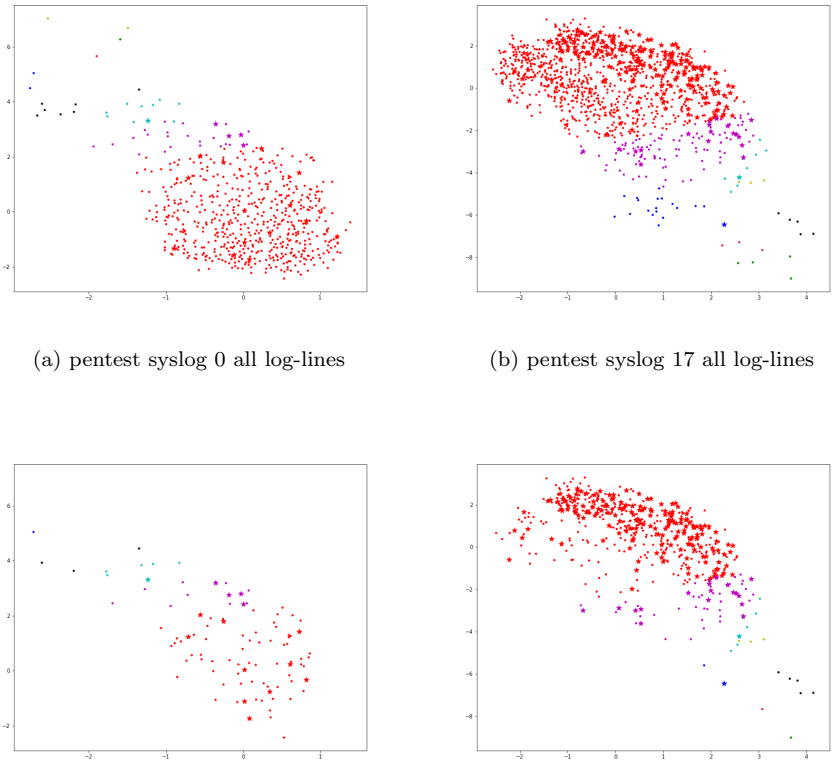


(a) 2D clustering



(b) 3D clustering

Fig. 4.4: Clustering syslogs by Normalized Compression Distance in 2D (a) and 3D (b). The dots denote normal TU/e logs and the stars denote anomalous pentest logs. Each colour denotes a different cluster.



(a) pentest syslog 0 all log-lines

(b) pentest syslog 17 all log-lines

(c) pentest syslog 0 only anomalous log-lines

(d) pentest syslog 17 only anomalous log-lines

Fig. 4.5: NCD clustering on individual log-lines of pentest syslog 0 in (a) and (c) and pentest syslog 17 in (b) and (d). A clustering of all log-lines is shown in (a) and (b) and a clustering of only the anomalies (true anomalies (stars) and false positives (dots)) according to the word-based BEM are shown in (c) and (d).

4.1.4 Using TF-IDF for Anomaly Detection and Revisiting the Attention Layer

Anomaly Detection with TF-IDF

The TF-IDF scores of the "$\langle EOS \rangle$" characters are plotted in Fig. 4.6. It is important to note that in this specific case the TF-IDF score is equal to the TF of "$\langle EOS \rangle$" within the range [0,1], as explained in Section 3.5.4. It can be seen that this metric seems to be able to get a good separation between normal logs and a good number of pentests/fault logs. Because the TF-IDF scores of the "$\langle EOS \rangle$" character seem to have some predictive power, we decided to use them as a new dimension to our inter-log anomaly detection plots.

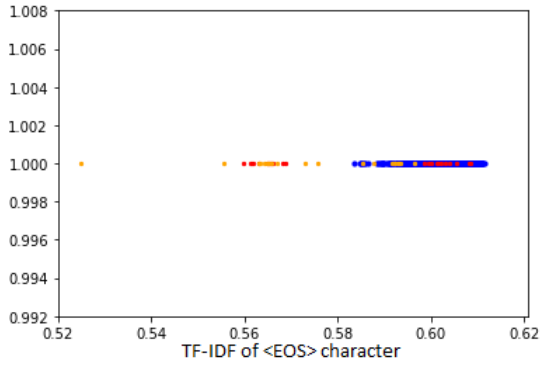
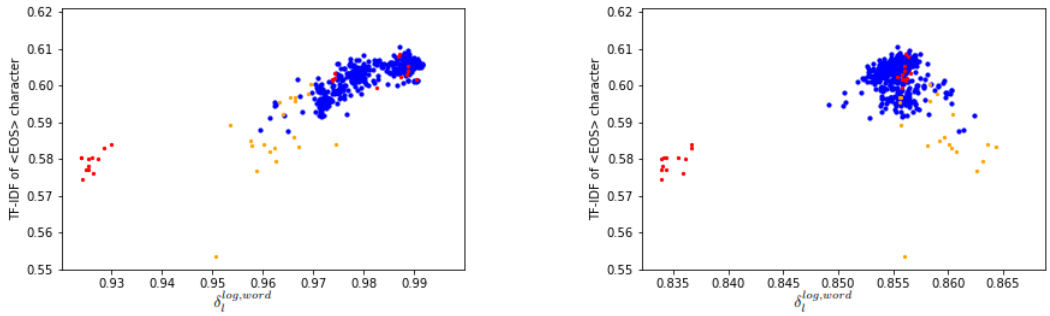


Fig. 4.6: The TF-IDF scores of the "$\langle EOS \rangle$" character for TU/e logs, pentests and fault logs.

Attention layer for word-based BEM

Even though the A-EM had bad results, we decided to revisit the attention layer for the word-based BEM, because it could give us an avenue for interpretability. The attention layer was added to the word-based BEM LSTM, creating the A-BEM. The results are set against the TF-IDF scores for the logs with and without attention layer in Fig. 4.7. We see that the TF-IDF scores of the "$\langle EOS \rangle$" character indeed have a positive impact on the potential performance of the support vector machine, as most of the fault logs would overlap with the TU/e logs if only the $\delta_l^{log,word}$ scores were used. As for the attention layer, we see that the fault logs have higher scores using the A-BEM versus the BEM compared with the TU/e logs. We also see that the pentest logs that are similar to the TU/e logs seem to form a vertical grouping using the A-BEM, while with the BEM they display no such grouping. While training a support vector machine with non-linear decision boundary might provide a better performance on the A-BEM due to the vertical grouping of pentests, this would require an overfitting of the data which we are not sure will transfer well when used on general data. While the performance of the A-BEM seems better than the A-EM, we still decided to use the BEM rather than the A-BEM because the A-BEM assigned the highest $\delta_l^{log,word}$ scores to the fault logs while we would like the model to assign lower scores to logs with faults in them. The A-BEM is also really computationally expensive, so we would rather train a slightly weaker model than one that takes multiple days to train.



(a) BEM (b) A-BEM

Fig. 4.7: The BEM against the A-BEM with anomaly scores set against the TF-IDF scores for the "$\langle EOS \rangle$" character for TU/e logs, pentests and fault logs.

4.1.5 Experiments with Hackathon Logs and Removal of Non-Verbose Pentests

At this point in our work a hackathon was held at the TU/e where the goal was for the students to hack into the STEP system. The logs from the hackathon were compiled into the hackathon dataset and from this pont onward we gained access to this dataset. We now have access to the complete dataset described in Section 3.3.1 for all our experiments from here onwards. We also found out that there are two kinds of pentest logs. Pentest syslogs that seem to have a verbosity mode on and have large file sizes, these are the ones that are easy to distinguish from TU/e logs because none of the TU/e logs use this verbosity mode. And pentest syslogs that do not have this verbosity mode on, which are the ones that are challenging to distinguish from TU/e logs. The problem that we are facing, is that we do not know whether or not the non-verbose pentests are actually of hack attempts, it is possible they were created when the pentesters were testing the STEP USB sticks and do not constitute fraud. Hence, we made the decision to remove the non-verbose pentests. We decided to keep the verbose pentests because the usage of the verbosity mode might be an indicator of a hack attempt, just not a very useful one if we want to find anomalous log-lines within a normal syslog. The issues with this decision are that the verbose pentests already were easily distinguishable because of the fact they were using the verbosity mode, so we conclude that the pentests do not tell us anything interesting because of the verbosity mode and that we cannot be sure that the non-verbose pentests are actually of hack attempts. Because of this we also decided not to run any intra-log anomaly detection experiments anymore because we have doubts about the degree to which these two logs—pentest 0 as the non-verbose syslog and pentest 17 as the verbose syslog—actually mirror a log that we would find from an actual hack attempt.

Decision Boundary with Hackathon Logs

We now recompute the decision boundary for the complete dataset including the hackathon logs. The support vector machine is used with "kernel" set to "rbf", "class_weights" set to "8" for an anomaly and "1" for the normal class with the "C" parameter set to "2" and "gamma" set to "auto". The decision boundary for all verbose pentests, fault logs, hackathon logs and the 636 normal test TU/e logs is given in Fig. 4.8 and the performance metrics in Table 4.4. As it can be

seen this decision boundary works extremely well with a balanced accuracy of 0.9914, finding all anomalous logs and returning only 11 false positives out of 636 logs. The hackathon logs seem to have similar scores to the fault logs. While the performance seems extremely good, we will see in Section 4.2.2 that these results will not scale as well to the general case when the model is fined-tuned and some bias is removed.

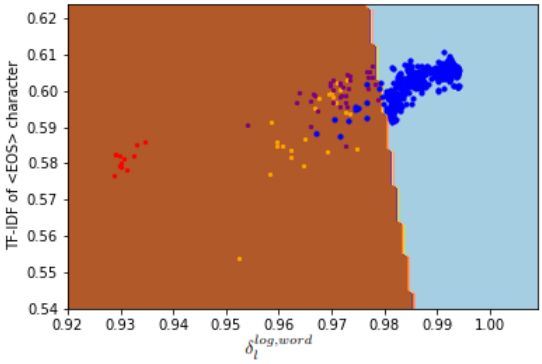


Fig. 4.8: The decision boundary given by the support vector machine on all log types. It is possible to divide all anomalous logs with minimal false positives.

Balanced Accuracy	0.9914
F1 Score	0.9272
Confusion Matrix	[[625 11] [0 70]]

Table 4.4: Performance metrics for Fig. 4.8. All 70 anomalies are found, with only 11 false positives.

4.1.6 Deployment

The base trained model (BEM) can be included with every copy of the STEP system. This can be used as a basis and should be retrained when new data comes in. A subset of sessions that have high anomaly scores should be used to continue training the model on, here it is assumed that sessions with high anomaly scores do not have any or very few anomalous log-lines. It is also possible to get (partially) labelled anomalous sessions. These could be used to fine-tune the model in the future, for example by training a supervised model once enough data has been gathered.

4.1.7 Summary Anomaly Detection Experiments and Results

We tested the EM, BEM, A-EM and A-BEM introduced in Section 3.5.2 using both character and word-based methods. We found that the character-based EM was the weakest model and that the added attention layer in the character-based A-EM severely decreased the performance of the model. An isolation forest was also considered for anomaly detection, but its performance was not good enough for us to consider it as a viable alternative to the EM-based models. Instead, the isolation forest was used as a baseline for intra-log anomaly detection on the EM-based models. The character-based EM and the character-based BEM performed similarly with both models

not really outperforming the other in different aspects. We still decided to go with the BEM because of historical evidence that the BEM outperforms the EM from the original paper. We then tested the BEM with the word-based method and found that it outperformed the other character-based methods tested thus far. For increased training speed and because we got good results, we decided to look further into word-based methods rather than character-based methods. We also tested a support vector machine in order to set a good general decision boundary and got decent results. Next we tried to cluster syslogs and log-lines with the normalized compression distance for inter and intra-log anomaly detection, but found that neither clustering produced good results. We then decided to try clustering on TF-IDF score, and we found out that we can use the TF-IDF score of the "$\langle EOS \rangle$" character of each log-line as a decent anomaly score. However we later found out that the TF-IDF score was actually computing only the TF score because of special circumstances with the "$\langle EOS \rangle$" character, therefore it will not be a good method if used in this way outside this very specific scope. We tried the attention layer again on the A-BEM and we got better results than the A-EM, but we still decided not to use the A-BEM because it was assigning high anomaly scores to the fault logs and its computation time was very high. Finally we got access to the whole dataset and we decided to remove non-verbose pentests because we were not sure if they actually resulted from hack attempts. Because of this we are not sure to what extent the pentest 0 and pentest 17 log used for intra-log anomaly detection mimic true hack attempts. Because the verbose pentests are actually anomalous—because using verbosity mode is already an anomaly—we decided to keep them. Then we tried to create a decision boundary with the support vector machine again, and we achieved very good results, partly because of the removal of the non-verbose pentests. However, we will see in the next section that this model does not scale as well after improving the model further. Table 4.5 has a summary of all the methods used.

Name	Sub-types			Reason	Limitations
Event-based LSTM Autoencoder				Find collective anomalies	1) Input shape mismatch with new logs 2) Drain errors decrease performance 3) Logs too complicated for this method to work well
LSTM	Character-Based	Unidirectional	No Attention (EM)	Find point and collective anomalies Vocabulary is constant	Performance is sub-optimal without bidirectional LSTM
			Attention (A-EM)	Attention can improve performance and interpretability	Attention decreases performance and significantly increases training time
		Bidirectional	No Attention (BEM)	Bidirectional LSTM has better performance	The output of the model was not compatible with SHAP DeepExplainer
	Word-Based	Bidirectional	No Attention (BEM)	Word-Based method adds additional options for interpretability (SHAP values)	Performance is slightly worse than the character-based model. Vocabulary becomes larger as corpus increases, leading to memory issues
			Attention (A-BEM)	Attention can improve performance and interpretability	Attention decreases performance and significantly increases training time
Isolation Forest				Fast model that works reasonably well out of the box	Performance is not very good
Normalized Compression Distance				Can cluster raw log-lines using only their binary values, always usable	Results are mediocre
TF-IDF				Better metric for which words are important Provides more clustering options	In our case only the TF value is used

Table 4.5: Overview of different methods used

4.2 Interpretability and Model Optimisation

It was decided to do a simple visualization of the model rather than more in-depth interpretability options because none of the interpretability methods discussed in Section 3.6.1 looked promising for the model. A small case study was held with experts from Carapax IT and further improved the model based on the feedback from the case study.

4.2.1 Visualization

The predictions of the model were visualized by generating Excel files for every syslog. These Excel files contain the $\delta_l^{log-line,word}$ for every log-line and the δ_i^{word} for every word is visualized using a colour mapping from red to green where red is closer to 0 and green is closer to 1. An example of the visualization in Excel is given in Fig. 4.9.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		original	average probability	0	1	2	3	4	5	6	7	8	9	10	11
125	123	Nov 5 17:5 kernel: [0.000(	0.997639835	kernel	ioapic	apic	version	address	xfec	gsi					
126	124	Nov 5 17:5 kernel: [0.000(	0.99937135	kernel	acpi	int	src	ovr	bus	bus	irq	global	irq		
127	125	Nov 5 17:5 systemd[1]: Star	0.67756623	systemd	started	set	cpu	frequency	scaling	governor					
128	126	Nov 5 17:5 kernel: [0.000(	0.997510791	kernel	acpi	used	override								
129	127	Nov 5 17:5 kernel: [0.000(	0.987516761	kernel	using	acpi	madt	smp	configurat	information					
130	128	Nov 5 17:5 kernel: [0.000(	0.99736011	kernel	acpi	hpet	base	xfed							
131	129	Nov 5 17:5 kernel: [0.000(	0.999803543	kernel	smpboot	allowing	cpus	hotplug	cpus						
132	130	Nov 5 17:5 kernel: [0.000(	0.995948792	kernel	registered	nosave	memory	mem							
133	131	Nov 5 17:5 systemd[1]: Rea	0.791000664	systemd	reached	target	host	network	name	lookups					
134	132	Nov 5 17:5 grub-common[1	0.017711299	grub	common	recording	successful	boot	grub						
135	133	Nov 5 17:5 grub-common[1	0.664594769	grub	common	done									
136	134	Nov 5 17:5 systemd[1]: Star	0.838757217	systemd	started	lsb	record	successful	boot	grub					
137	135	Nov 5 17:5 kernel: [0.000(	0.977144003	kernel	mem	xdfffffff	available	pci	devices						
138	136	Nov 5 17:5 kernel: [0.000(	0.997169197	kernel	booting	paravirtua	kernel	bare	hardware						
139	137	Nov 5 17:5 kernel: [0.000(	0.999971807	kernel	clocksourc	mask	max	cycles	max	idle					
140	138	Nov 5 17:5 kernel: [0.000(	0.99984026	kernel	random	get	random	bytes	called	start	kernel	crng	init		
141	139	Nov 5 17:5 kernel: [0.000(	0.999899268	kernel	setup	percpu	cpus	cpumask	bits	cpu	ids	node	ids		
142	140	Nov 5 17:5 rsyslogd: imuxsc	0.932780623	rsyslogd	imuxsock	acquired	unix	socket	run	systemd	journal	syslog'	systemd		
143	141	Nov 5 17:5 rsyslogd: rsyslog	0.99614954	rsyslogd	rsyslogd's	changed									
144	142	Nov 5 17:5 systemd[1]: Star	0.997428596	systemd	started	service									
145	143	Nov 5 17:5 rsyslogd: [origin	0.998996794	rsyslogd	origin	software	rsyslogd	swversion	pid	info	http	www	rsyslog	com	start

Fig. 4.9: An example of how the visualization of the model looks like in Excel with an anomaly in red.

4.2.2 Expert Feedback and Model Optimisation

Feedback

The visualization was shown to the technical expert and tester from Carapax IT and they provided feedback on the visualization. The textual feedback received from the experts is shown in Appendix C. The technical expert's opinion was that the visualization helped him to find faults faster by putting emphasis on the problem areas within a log. They noted that some faults were not found because they were collective anomalies. Finally, they said that seeing the average anomaly scores in combination with the context of the log-lines helps them the most in finding faults faster. Additionally, when examining the visualization in more detail it was found that several new STEP log-lines were not being extracted to log keys satisfactorily, this caused a large amount of functionally the same log-lines to not be removed correctly. Since it was decided to use unique reduced log-lines only, having one log-line that is not unique could skew the results of the anomaly scores for the newer logs which always contained those new STEP log-lines as these log-lines have a very low anomaly score. In fact in Fig. 4.8 there is a good separation between anomalous and normal logs, however all hackathon (purple) and a subset of the fault (yellow) logs contain these new STEP log-lines which skews their log-line anomaly score session average further to the left than they should be. While it will likely reduce performance it is better to resolve this issue so as to better model the reality.

Since it was decided to find point anomalies rather than collective anomalies, it will be impossible for the model to find these collective anomalies, therefore expanding the model to find collective anomalies will fall under future work.

4.2.3 Model Optimisation Based on Feedback

After the case study, it was decided to further optimize the model. The technical expert and tester of Carapax IT were asked to mark false positives log-lines with anomaly scores of 0.2 or lower. This provided a list of 73 false positives. The training phase was also scaled up and the model is now being trained on 1000 logs rather than just 100 logs. Training on more logs will increase the anomaly scores for normal lines and should push non-anomalous logs further to an anomaly score of 1.

Fine-tuning on False Positives

Because there will always be log-lines that will be incorrectly indicated as anomalous—which decreases the usability of the model—it is in our interest to add a way to fine-tune the model on these false positive log-lines. The idea we came up with to fine-tune the model is that we generate a couple of new logs which contain false positives identified by the experts. Those logs will automatically get trained on by the model in order to give those false positive log-lines a higher $\delta_l^{log-line,word}$ score. The BEM was first trained on two logs (one for each visualized log that the experts looked at) containing all false positives identified by the experts. However, a relatively large change was noticed in absolute average changes between $\delta_l^{log-line,word}$ scores before and after the fine-tuning, disregarding score changes in any log-lines that were trained on during fine-tuning. It is possible that the model might be overfitting on the false positives. In order to avoid overfitting on the false positives a new log of 700 log-lines was created which contains the 73 false positives and 627 normal lines randomly sampled without replacement from the 1000 logs in the training set. The reason to create this kind of log is to give the model a log that is similar to a TU/e log but with the false positive log-lines added such that it hopefully does not overfit the model on the false positives anymore. The order of these 700 log-lines is randomized to avoid bias for the false positives.

Batch Size

Originally a batch size of 512 was used, which is on the high side. Because the fine-tuning on false positives logs with only the 73 false positives gave rather large absolute changes in log-line anomaly scores for log-lines that were not trained on during the fine-tuning step, it was decided to tune the batch size to decrease the differences in the rest of the model. Large batch sizes can lead to decreased ability to generalize in models that use stochastic gradient descent (Keskar et al., 2016). Hence it was decided to change the batch size to 32 which is the most common safe option for a batch size. While a number of different batch sizes were tried before arriving at 32 it is still an open question whether a batch size of 64, 128 or 256 performs better than 32 for the model.

Early Stopping

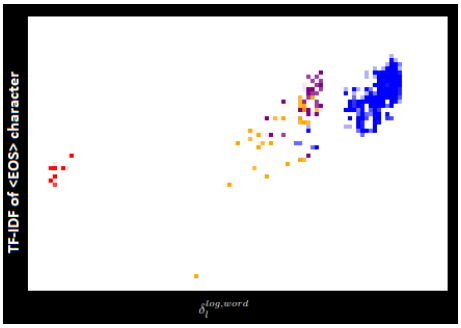
Now that 1000 logs are being used in the training phase it is important that the model is not overfitting. Early stopping was implemented to monitor the validation loss with a “min_delta” of “0.0005” and “patience” set to “3”. Early stopping on loss gave worse results than early stopping on validation loss.

Adjusting Tokenizer

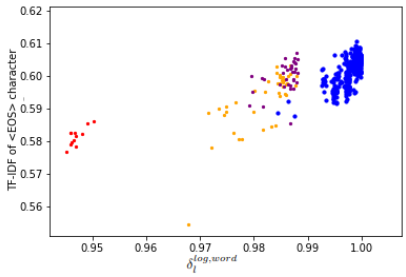
It was noticed that stop words were getting high anomaly scores and single characters were still a part of the vocabulary. Stop words can skew the anomaly scores up since they often have high anomaly scores and are not indicative of anomalous behavior. The tokenizer was modified to remove stopwords and single characters.

Effects of Fine-tuning

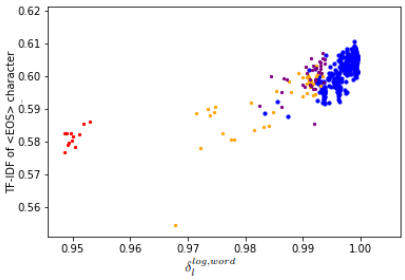
The new model that is trained on 1000 logs is used, with a batch size of 32 and early stopping set to on. The effects of fine-tuning and the new tokenizer are shown in Fig. 4.10. Subfigure (a) has a very nice separation between normal and anomalous cases. However once the model is fine-tuned this nice separation disappears and the performance decreases. This seems to indicate that the false positives that were removed are actually a large part of the reason that this nice separation happened in the first place. It is possible that these are new log-lines that were added in newer logs and it reveals a weakness in the model where any log that is newer than the version of STEP used for training might be automatically ranked as anomalous, even if this is not necessarily the case. An important question for further work is whether or not the model has to be retrained on new normal logs for every new version of STEP or whether or not it is fine to continue training the existing model on logs from new versions of STEP.



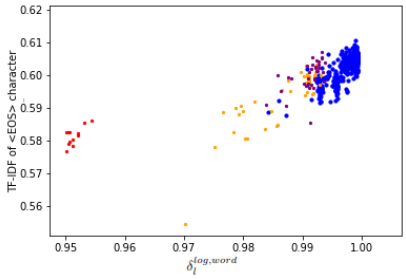
(a) No fine-tuning, old tokenizer



(b) No fine-tuning, new tokenizer



(c) Fine-tuning, old tokenizer



(d) Fine-tuning, new tokenizer

Fig. 4.10: Effects of fine-tuning and using the new tokenizer without stopwords and single characters. Fine-tuning decreases the performance of the model. Note that the original high resolution version of Subfigure (a) with proper axis labeling was lost and this image is the only one still available.

4.2.4 Adjusting Drain

Because of the feedback from the technical expert from Carapax IT that several STEP log-lines were not being extracted to log keys satisfyingly, it was decided to adjust Drain's similarity threshold from 0.5 to 0.3. This change proved to be enough to correctly extract the log keys of the new STEP log-lines. It is important to note that Drain will extract all log keys more aggressively, hence any other log-line might have changed too. Fig. 4.11 shows what happens with the anomaly scores when Drain's similarity threshold is set to 0.3. From the performance metrics in Table 4.6 it is observed that the performance decreases when Drain's similarity threshold is lowered to 0.3. A large amount of hackathon and fault logs now become non-separable from the non-anomalous logs. While it seems like a bad result, it was expected that the hackathon and a subset of the fault logs would become more normal as the bias induced by the new STEP log-lines is removed. These results should, however, give a more accurate representation of how the model will work in a real environment.

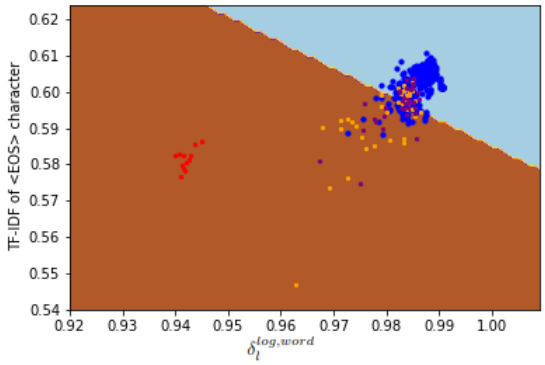


Fig. 4.11: As expected removing the error caused by the new STEP log-lines pushes the hackathon and fault logs further towards normal instances.

Balanced Accuracy	0.7505
F1 Score	0.5397
Confusion Matrix	[[586 50] [37 51]]

Table 4.6: Performance metrics for Fig. 4.11. Performance decreases significantly by adjusting Drain's similarity threshold.

4.2.5 Summary Interpretability Experiments and Results

In this section we looked into interpretable machine learning models to use for the BEM and model optimisation based on expert feedback. In Section 3.6 it was found that interpretability using SHAP values and Seq2Seq-Vis were not possible because those methods do not work with softmax output. The most promising interpretability method that was found for the BEM is some form of attention mechanism, however because the attention layer had a very negative effect on the model's predictions this path was not continued. Instead a visualization in Excel was created using the softmax values predicted for each word in the log-line. A case study was conducted where experts from Carapax IT tried to find faults and hack attempts in a fault and hackathon

log. The experts found the visualization useful and potentially a time saver when going over logs, but they noted that some anomalies are collective anomalies which cannot be found with this method. Based on the feedback of the experts the model was further improved. The experts had compiled a list with false positives which the model should be able to learn without compromising the integrity of the existing model weights as much. To combat degrading the model too much, the batch size was changed from 512 to 32, early stopping was implemented, the tokenizer was adjusted by removing stopwords and single characters and finally a new log was created with false positive log-lines randomly mixed with normal TU/e log-lines sampled without replacement from the training set. While the performance decreased, the new settings should mirror the real environment better. Furthermore, the Drain algorithm was set to generate log keys more tightly to combat similar log-lines not being reduced to unique instances and introducing bias into more modern logs. While the performance continues to decrease with this improvement, this should still mirror the real environment better. Further research will be needed to test the performance of this new model in a real environment as well as the performance for finding point anomalies on log-line level with the new method.

Chapter 5

Conclusions and Lessons Learnt

We conclude the thesis with our conclusions, summary of the main contributions, overview of limitations and future directions and well as recommendations to the company how to further develop the anomaly detection system.

5.1 Conclusions

Possible machine learning methodologies for anomaly analytics to use for finding faults and fraud attempts in a secure testing environment have been explored. We discovered that using an event model with bidirectional LSTM (BEM) to predict the next word in a sentence gives the best results. The model was trained on logs that are considered to only contain normal behaviour and so the model can reconstruct normal behaviour but will struggle with abnormal behaviour. If the model reconstructs abnormal behaviour the model will have a high reconstruction error, in other words, the model will have a low probability for a specific word in a specific context, these probabilities were used to identify anomalous logs and log-lines. Finally, it was concluded that there were no interpretable anomaly alert processing solutions that were suitable for the model implementable within the scope of this study, hence a simpler visualization of the output was created in Excel.

5.2 Main Contributions

We determined that Machine learning for anomaly detection (jailbreak detection) in secure testing environments is a new research sub-domain within anomaly detection and fraud detection machine learning research after performing a thorough search of the relevant literature, which yielded no related papers. An initial exploration was performed of the problem space and we found that techniques from the natural language processing field proved moderately successful in finding anomalous sessions and log-lines. In particular using an event model with bidirectional LSTM (BEM) to predict the probabilities of all words in the vocabulary for the next word in a sentence. Then, the probability of the actual word in a sentence can be used as a word-based anomaly score; words that are known in this specific order by the model will then be considered normal and words that are not known in this specific order will be considered anomalous. An assumption that is made is that the model is only trained on normal logs hence any anomalous lines will not appear in these logs and these anomalous lines can be detected by the model. Furthermore, we tried to apply interpretable anomaly alert processing techniques but it was

concluded that none of the techniques would work on the model within the scope of our research, some kind of attention mechanism seems to be the most promising for future research. Instead a visualization of the model was created in Excel and we conducted a case study of the model with the technical expert and the tester from Carapax IT. The technical expert and the tester thought the visualization was useful and might help them save time when searching for anomalies in logs in the future. Based on the feedback of the experts the best performing implementation of BEM was updated again with a fine-tuning mechanism and some parameters were tweaked. The improvements to BEM should mirror reality better but the performance of BEM is less than BEM without the improvements. Future research should look further into improving the new model and implementing it into a real environment.

5.3 Limitations and Future Work

All of the abnormal logs have been obtained through controlled experiments with students and pentesters and the fault logs were only those that were reported by the educational institutions. Many logs might also be from an older version of the STEP system, so those logs might have differences in STEP related log-lines that have changed over time. An open question is to what extent this influences the results of the model. While most of the abnormal logs were found with BEM, was this because these logs were abnormal or because they contained normal log-lines that were not in the training set. These limitations will make it more challenging to keep the system up-to-date, as every time a new version of the STEP system is rolled out there potentially is a need to gather a large amount of new normal logs to train the model on from scratch. This leads to another open question of whether the model should be retrained from scratch with every new version of STEP or if the existing model should be updated, does it degrade the predictive capabilities if logs from different versions of STEP are used or does it not matter.

There are only two labeled logs from the pentesters which were labeled by the expert, the pentester logs were also using some kind of verbosity mode in the logs which makes it impossible to accurately gauge if these logs would be found without verbosity mode on. Since there are only two logs paired with the verbosity mode issue it's hard to decisively conclude anything from these logs about the model performance.

The expert told us that there were collective anomalies that the model did not find. Because the model can currently only find point anomalies it is future work to extend this to collective anomalies. This might be done by adding tiering to the BEM model, i.e. building the T-BEM or TA-BEM models. However, the model is currently using the fact that we are only looking for point anomalies in its fine-tuning mechanism.

The anomaly scores can be updated to use the negative log-likelihood instead of using the mean over the probabilities for individual tokens.

The AUC or pAUC can be used as a better performance metric for future iterations of the methods.

There is no absolute confirmation that the TU/e dataset only contains normal data, if the TU/e dataset does contain anomalous data that is of interest to the company, then those anomalies might be impossible to find by the model.

5.4 Lessons Learnt and Recommendations

What we learned is that building a model of normal behaviour—and using this model to identify anomalous behaviour outside of the model—seems promising for finding anomalies in custom Linux log data. However, because the data was gathered in such a way that we cannot be sure that TU/e logs and logs with fraud or faults are truly comparable, we do not yet know how well our method will generalize to a live environment. Deep learning techniques from natural language processing seem promising to build a good normal model of sentences within Linux logs. Vocabulary of Linux logs differs from natural language, several rules of natural language processing should be tweaked such that they work optimally for log data. Many log-lines are generated with the same code which contains variables, it is important to filter variables out and only keep the original log-line with a stand-in for variables, as excessive variable values can easily fill the vocabulary with bogus words and increase the amount of unique lines.

The recommendation for the company is to implement the solution into STEP and experiment using the predictions in practice. Care must be taken that the model translates correctly to a real environment. To get better data for testing, it is our recommendation to hold a hackathon during a live exam where security students—who do not have to actually take this exam—attempt to circumvent the STEP system. In this case, the logs from normal students and the hacker students can be better compared to each other and this should prevent issues where the logs were generated under vastly different circumstances. An open question is whether or not the model has to be retrained for every new version of the STEP system or that the model continues to be trained in an on-line fashion on new logs from the educational institutions. If the model is trained on many logs at the same time, it will likely take hours or days depending on the hardware available. It might also be worth it to develop a new method to find collective anomalies or implemented the TA-EM or the TA-BEM.

Bibliography

Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*, page 1. ACM, 2018. 8, 36

André Carrington, Paul Fieguth, Hammad Qazi, Andreas Holzinger, Helen Chen, Franz Mayr, and Douglas Manuel. A new concordant partial auc and partial c statistic for imbalanced data in the evaluation of machine learning algorithms. *BMC Medical Informatics and Decision Making*, 20, 12 2019. doi: 10.1186/s12911-019-1014-6. 25

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009. 9

R. Cilibrasi and P.M.B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, Apr 2005. ISSN 0018-9448. doi: 10.1109/tit.2005.844059. URL http://dx.doi.org/10.1109/tit.2005.844059. 40

Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006. 24

Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017. 1

Aarish Grover. Anomaly detection for application log data. 2018. 8, 34

Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. An evaluation study on log parsing and its use in log mining. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 654–661, 2016. 32

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 33–40. IEEE, 2017. 31

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. 19

Richard M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*, page 416–429, NLD, 1992. North-Holland Publishing Co. ISBN 044489747X. 17

Alexander Kent. *Cyber security data sources for dynamic network research*, pages 37–65. 05 2016. ISBN 978-1-78634-074-0. doi: 10.1142/9781786340757_0002. 36

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. 57

Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. ISBN 0-201-89685-0. 18

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991. 19

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016. 12

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008. 18

Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):3, 2012. 18

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017. 42

Donna Katzman McClish. Analyzing a portion of the roc curve. *Medical Decision Making*, 9(3): 190–195, 1989. 25

Adam Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584. IEEE, 2007. 34

Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. 25

Anand Rajaraman and Jeffrey David Ullman. *Data Mining*, page 1–17. Cambridge University Press, 2011. doi: 10.1017/CBO9781139058452.002. 40

Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10:e0118432, 03 2015. doi: 10.1371/journal.pone.0118432. 25

Colin Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 2000. 27

H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush. Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *ArXiv e-prints*, 2018. 42

Wanhua Su, Yan Yuan, and Mu Zhu. A relationship between the average precision and the area under the roc curve. pages 349–352, 09 2015. doi: 10.1145/2808194.2809481. 25

David Tax. One-class classification; concept-learning in the absence of counter-examples. 01 2001. 16

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 21

Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132, 2009. 34

Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '10, pages 121–130, Piscataway, NJ, USA, 2019. IEEE Press. doi: 10.1109/ICSE-SEIP.2019.00021. URL https://doi.org/10.1109/ICSE-SEIP.2019.00021. 32, 36

Appendix A

Glossary

Domain Terms

- **STEP System:** (Secure Testing Environment Protocol) is a Linux based operating system that is loaded onto a USB stick.
- **USB stick:** In the context of this thesis this always refers to a USB stick with the STEP System loaded onto it.
- **Session:** All logs that have been generated by the STEP system for a single USB stick entered into a student machine from USB startup to shutdown.
- **Log-line:** A single line from any of the logs generated during a session.
- **Fault:** A failure in software and/or hardware which occurs when a computer system is running the STEP System. The cause of a fault is logged in Linux and/or STEP logs, but might not be easy to find for a layperson. Finding what causes faults in the STEP system helps improving the system in newer iterations.
- **Fraud:** A malicious attack on the STEP system by a bad faith actor. When a malicious attack on the STEP System succeeds this will allow the bad faith actor to cheat on an examination. Fraudulent log-lines are log-lines which show or give evidence for the claim that a fraudulent action has taken place.
- **Anomaly:** A pattern or an instance in the data which does not conform to expected behaviour. An anomaly in the STEP System may denote either a fault or fraud but can also be neither.
- **TU/e dataset:** A set of 3639 sessions generated during real use at the TU/e. Considered to only contain non-anomalous data.
- **Pentest dataset:** A set of 27 sessions generated by a company trying to hack the STEP System. Considered to contain anomalous data.
- **Fault dataset:** A set of sessions with known faults.
- **Hackathon dataset:** A set of sessions from the TU/e hackathon who attempted the same hacks as the pentesters.

- **Inter-log anomalies:** Anomalies on log level i.e. inter-log anomaly detection finds anomalous logs.
- **Intra-log anomalies:** Anomalies on log-line level i.e. intra-log anomaly detection finds anomalous log-lines.

Abbreviations

- **STEP:** Secure Testing Environment Protocol
- **TU/e:** Technische Universiteit Eindhoven
- **NLP:** Natural Language Processing
- **LSTM:** Long Short-Term Memory
- **RNN:** Recurrent Neural Network
- **TN:** True Negative
- **TP:** True Positive
- **FN:** False Negative
- **FP:** False Positive
- **AUC:** Area Under the Curve
- **pAUC:** Partial Area Under the Curve
- **CRISP-DM:** CRoss-Industry Standard Process for Data Mining
- **ASCII:** American Standard Code for Information Interchange
- **BOS:** Begin Of Sentence
- **EOS:** End Of Sentence
- **HDFS:** Hadoop Distributed File System
- **BGL:** Blue Gene/L
- **LANL:** Los Alamos National Laboratory
- **EM:** Event Model
- **BEM:** Bidirectional Event Model
- **T-EM:** Tiered-Event Model
- **T-BEM:** Tiered-Bidirectional Event Model
- **A-EM:** Attention-Event Model
- **A-BEM:** Attention-Bidirectional Event Model
- **TA-EM:** Tiered Attention-Event Model
- **TA-BEM:** Tiered Attention-Bidirectional Event Model

- **SHAP:** SHapley Additive exPlanations
- **Seq2Seq-Vis:** Sequence To Sequence Visualization
- **NCD :** Normalized Compression Distance
- **TF-IDF:** Term Frequency-Inverse Document Frequency
- **MAC address:** Media Access Control address
- **PCI:** Peripheral Component Interconnect
- **USB:** Universal Serial Bus

Appendix B

Preliminary Experiments

B.0.1 Preliminary experiments with pentest logs

For the first experiments only the TU/e dataset and the pentest dataset were available.

Character-Based EM with Duplicates for Intra-Log Anomaly Detection

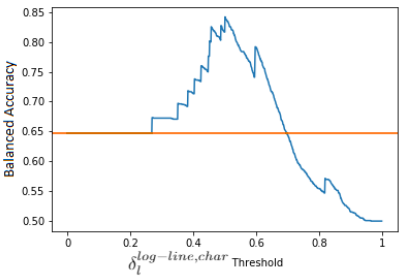
The labeled pentest 0 and 17 were used for evaluation. The character-based EM of 64 cells and a dropout layer of 0.2 has been trained. The model was trained on a set of 100 logs for 900 epochs, then on a new set of 100 logs for 200 epochs and finally on a set of 1000 logs for 20 epochs. The full logs were used for this without any duplicate log-lines removed. Intra-log anomaly detection is employed, the log-line anomaly score $\delta_l^{log-line,char}$ as defined in Section 3.5.2 is calculated for every log-line in pentest 0 and 17 and naive substring filtering is used as well to mark every log-line that contains a certain substring as defined in Section 3.5.4 as anomalous. The metrics are calculated by varying the $\delta_l^{log-line,char}$ threshold from 0 to 1 in 1000 increments of 0.001 and plotting the metrics for pentest 0 and 17 versus the $\delta_l^{log-line,char}$ threshold. Since certain log-lines are always tagged as anomalous due to naive substring filtering, this creates a baseline for the metrics as shown with the orange lines in Fig. B.1. The best performance based on highest balanced accuracy achievable on pentest 0 and 17 was with an $\delta_l^{log-line,char}$ threshold of 0.501 and 0.539 respectively and are shown in Table B.1. Since it is not known what the $\delta_l^{log-line,char}$ threshold will be that gives the best performance, it is a challenge to set one that works well in the general case on any log without labels available.

Character-Based EM without Duplicates for Inter-Log Anomaly Detection

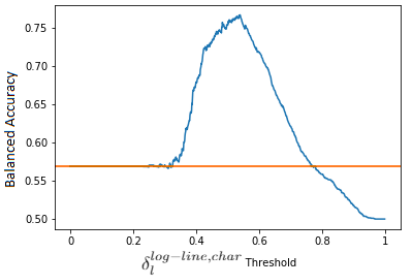
Next the character-based EM (with 64 cells and dropout layer of 0.5) was trained on 100 logs from the TU/e dataset with duplicate log-lines removed for 900 epochs, then on a new dataset of 100 TUE/e logs for 200 epochs and finally on a dataset of 1000 TU/e logs for 120 epochs. The predictor is validated against 1/3 of the input. Inter-log anomaly detection is employed,

	pentest 0	pentest 17
Balanced Accuracy	0.8419	0.7671
F1 Score	0.2481	0.4777

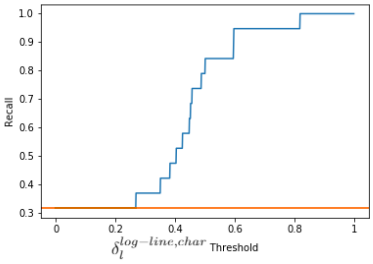
Table B.1: Performance metrics for character-based EM on pentest 0 and 17.



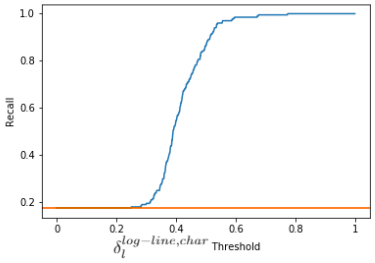
(a) Pentest 0 Balanced Accuracy



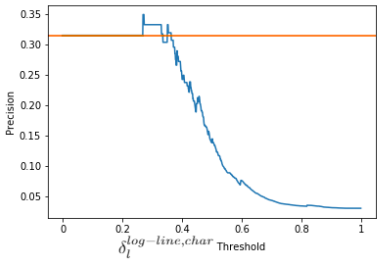
(b) Pentest 17 Balanced Accuracy



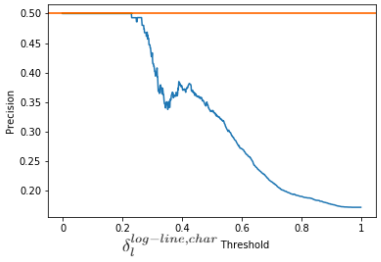
(c) Pentest 0 Recall



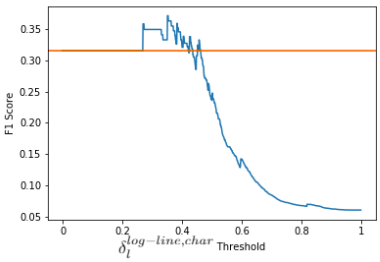
(d) Pentest 17 Recall



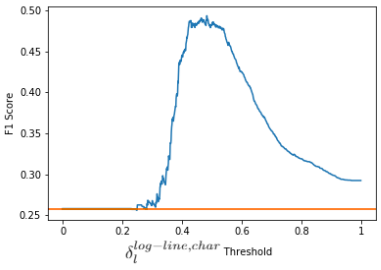
(e) Pentest 0 Precision



(f) Pentest 17 Precision



(g) Pentest 0 F1 Score



(h) Pentest 17 F1 Score

Fig. B.1: Different metrics values over the $\delta_l^{log-line,char}$ threshold values for pentest 0 and 17. The orange line denotes the baseline when only using the naive substring filtering method of 3.5.4.

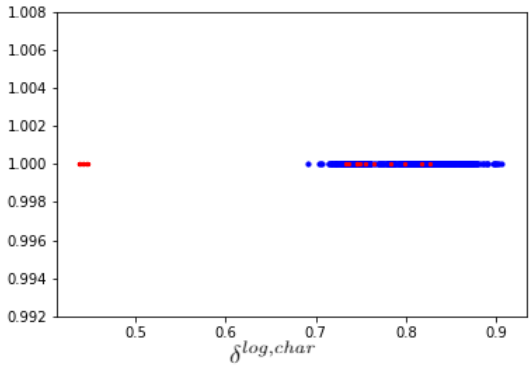


Fig. B.2: The log anomaly score $\delta^{log,char}$ for the TU/e dataset (blue) versus the pentest dataset (red) for the LSTM given by the character-based EM. A subset of pentest logs is easily separable from the TU/e logs.

the log anomaly score $\delta^{log,char}$, as defined in Section 3.5.2, is calculated for every TU/e syslog and pentest syslog which were not trained on, and naive substring filtering is used as well. The results—with TU/e syslogs in blue and pentest syslogs in red—plotted in one dimension by their log anomaly score are shown in Fig. B.2. The results show that one half of the pentest logs are easily separable from the TU/e logs while the other half are not.

Attention for Character-Based EM

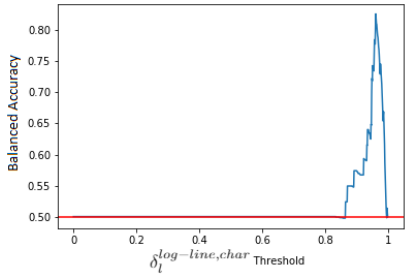
We tested the addition of an attention layer to the character-based EM model. The implementation of the attention layer from Datalogue [1] is used to create the A-EM. However, the A-EM—after training until the training loss stops decreasing and starts increasing—performs significantly worse (balanced accuracy of 0.31) than the EM (balanced accuracy of 0.74) and the A-EM has fairly low $\delta^{log,char}$ scores.

Character-Based BEM and Isolation Forest

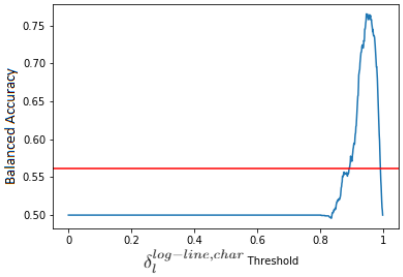
For the next experiment the EM was replaced with the BEM. The BEM was trained on 100 logs from the TU/e dataset for 250 epochs. From this point onward, naive substring filtering will not be used anymore to flag anomalies because we realized that there often are non anomalous log-lines that contain words like "error". A potential future direction for the naive substring filtering system is to let certain words add (negative) weights to the log-line anomaly score for log-lines containing those words, but this is left to future work. It was also decided to keep using logs with all duplicate log-lines removed, the reason for this choice is because some log-lines would repeat many times skewing the log anomaly score towards the log-line anomaly score of that log-line. While there might definitely be some value in keeping duplicate log-lines, for example when employing collective and contextual anomaly detection, since we are only using point anomaly detection removing all duplicates does not interfere with the usefulness of the method. This means our method is not fit to find any anomalies that are collective or contextual in nature, for example a large amount of repeating log-lines could indicate an anomalous log.

[1] https://github.com/datalogue/keras-attention/blob/master/models/custom_recurrents.py

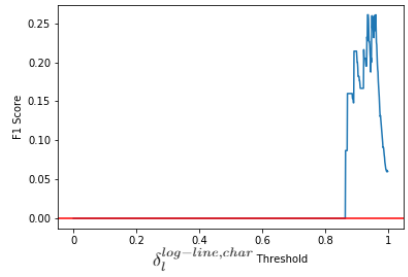
We also use an isolation forest as a new baseline to replace the naive substring filtering. The isolation forest model is fitted on the same 100 logs as the BEM using the character-based encoding of log-lines also used for the BEM. The isolation forest is used to predict whether log-lines or logs are anomalous using its default anomaly threshold of 0. The meta features of the isolation forest are set as follows: "behavior" is set as "new", "max_samples" is set as 100 and "contamination" is set as "auto". The isolation forest model is evaluated using the default isolation forest threshold 0 as the threshold between normal and anomalous log-lines. The evaluation metrics of the isolation forest model are shown as a red line baseline. In Fig. B.5 it is shown that both the BEM and the isolation forest can separate about half of the pentests sessions from TU/e sessions. In Fig. B.3 some evaluation metrics are shown for the BEM when varying the value of the $\delta_l^{log-line,char}$ threshold between 0 and 1 in 1000 steps of 0.001 on pentest 0 and 17. We see that the BEM outperforms the isolation forest. While the exact performance metrics were lost, we see that the EM is better on pentest 0 and the BEM is better on pentest 17. We decided to keep using the BEM because of its increased performance versus the EM in the original paper.



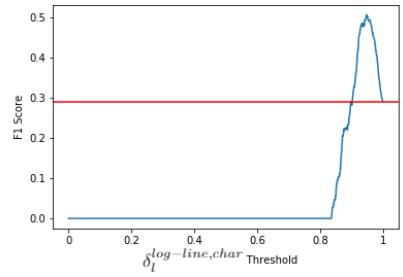
(a) Pentest 0 Balanced Accuracy



(b) Pentest 17 Balanced Accuracy

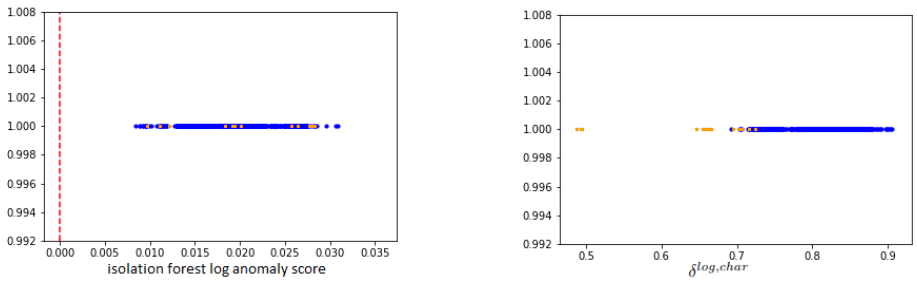


(c) Pentest 0 F1 Score



(d) Pentest 17 F1 Score

Fig. B.3: Different metrics values over $\delta_l^{log-line,char}$ threshold value of pentest 0 and 17 for the character-based BEM. The red line denotes the baseline from the isolation forest model with an anomaly threshold of 0. It can be seen that the character-based BEM outperforms the isolation forest for both pentests at an error threshold near 1.



(a) Isolation Forest

(b) Character-Based BEM

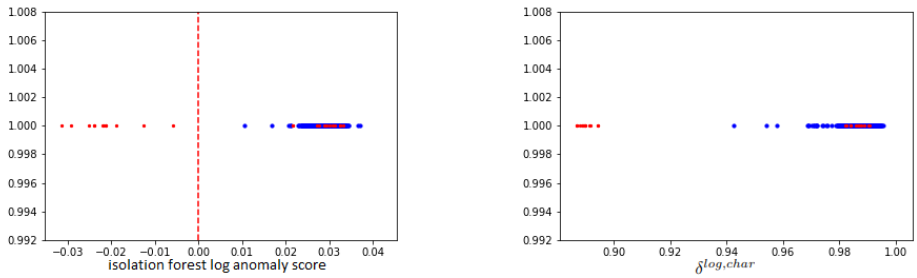
Fig. B.4: In (a) the isolation forest log anomaly score for the TU/e dataset (blue) versus the partial fault dataset (orange). In (b) the log anomaly score $\delta^{log,char}$ for the TU/e dataset (blue) versus the partial fault dataset (orange) for the character-based bidirectional LSTM. The vertical red dotted line in (a) denotes the anomaly score threshold of the isolation forest set at 0. The isolation forest cannot distinguish fault logs but the character-based BEM can.

Character-Based Fault Logs

Next we add the 13 logs from the partial fault dataset. Plots of the distribution for the partial fault dataset versus the TU/e dataset are shown in Fig. B.4. The isolation forest cannot distinguish between logs from the partial fault dataset and those from the TU/e logs at all. The BEM fares a lot better and can distinguish a large part of the fault logs without any mistakes. From the image it is clear that the character-based BEM could find all faults logs with a relatively low cost in false positives, as most TU/e logs have higher scores than all fault logs.

Anomaly Detection Threshold for Character-Based Models

Another question is whether we can train an algorithm to automatically set a good anomaly detection threshold for inter-log anomaly detection because the LSTM models need a variable anomaly detection threshold and do not use 0 as the standard anomaly detection threshold like an isolation forest model does. Since the $\delta^{log,char}$ scores might differ greatly based on how the model is trained, we need a more general applicable method. We decided to use a support vector machine for this purpose and train it on a set of known TU/e logs and pentest logs. Our hope is that this would allow the support vector machine to also correctly classify logs that are not part of the test set. The anomaly threshold is set manually by training the support vector machine with a linear kernel, the "C" parameter is set as "25" and the "gamma" parameter is set to "auto". The "class_weights" parameters are set as "1" for a false negative and "50" for a false positive. The advantage of this method is that the support vector machine can be retrained very quickly whenever the model is updated. The decision boundary given by the support vector machine for the graph of Fig. B.5 (b) is shown in Fig. B.6. We see that the decision boundary given is not perfect, as it includes two false positives. The pentests that are too similar to the TU/e logs are impossible to distinguish from TU/e logs with the current method.



(a) Isolation Forest

(b) Character-Based BEM

Fig. B.5: In (a) the isolation forest log anomaly score for the TU/e dataset (blue) versus the pentest dataset (red). In (b) the log anomaly score $\delta^{log,char}$ for the TU/e dataset (blue) versus the pentest dataset (red) for the character-based bidirectional LSTM. The vertical red dotted line in (a) denotes the anomaly score threshold of the isolation forest set at 0. Both the isolation forest and the character-based BEM have some success in distinguishing the pentest logs from the TU/e logs.

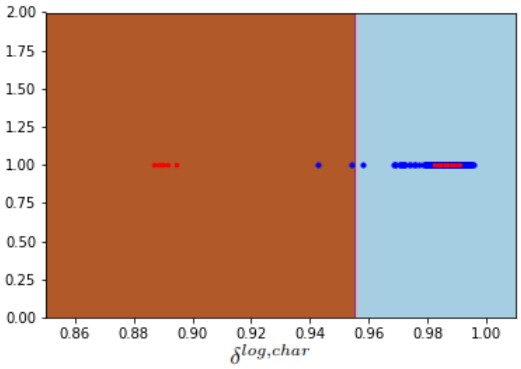


Fig. B.6: The decision boundary given by the support vector machine on the data of Fig. B.5 (b). It divides the TU/e logs and the pentest logs, but not perfectly.

Appendix C

Visualization Feedback

One filled out feedback form was received from the technical expert at Carapax IT. The tester of Carapax IT was asked to provide feedback as well, he did not have enough knowledge to mark false positives in the logs. The tester did not fill in the feedback form but gave other written feedback. He thinks the visualization makes reading the logs easier and he was 70% sure that the pattern of the alerts was right.

C.0.1 Example Feedback Received from the Technical Expert Original (Dutch)

Helpt de colortest om sneller fouten te vinden?:

Ja, wel om de aandacht op probleemgebieden te richten.

Waren er fouten die niet werden gevonden ("*averageprobability*" ≤ 0.2**) door het model?:**

Ja, maar dan moet het model met context zoeken: de stacktrace van de panic van de wifi-driver en in colortest_testlog_12.xslx een hoop verbose meldingen die er normaal niet zijn (bijvoorbeeld "xxx.timer: Timer elapsed."). Die zitten ongeveer tussen 0,2 en 0,4.

Welke elementen van de colortest helpen je sneller fouten te vinden?:

Vooral de average probablity en de volledige logregels (dus context).

Welke elementen van de colortest kunnen verbeterd worden om sneller fouten te vinden?:

---

geen idee ;)

Andere opmerkingen?:

-

C.0.2 Example Feedback Received from the Technical Expert Translated

Does the colortest [1] help to find faults faster?:

Yes, to put attention on the problem areas.

Were there faults that were not found ("*averageprobability*" ≤ 0.2) by the model?:

Yes, but then the model has to search with context: the stacktrace of the panic of the wifi-driver and in colortest_testlog_12.xslx a lot of verbose notifications that are normally not there (for example "xxxx.timer: Timer elapsed."). They are about 0,2 and 0,4[2].

Which elements of the colortest help you find faults faster?

Mostly the average probability and the complete log-lines (so context).

Which elements of the colortest can be improved to find faults faster?:

No idea ;)

Other remarks?

-

[1] Name of the visualization Excel sheet during testing
[2] Of average probability