Linköping University | Department of Computer and Information Science

Master's thesis, 30 ECTS | Datateknik

2023 | LIU-IDA/LITH-EX-A--23/040--SE

Clustering and Anomaly detection using Medical Enterprise system Logs (CAMEL)

–

Klustring av och anomalidetektering på systemloggar

Henrik Ahlinder
Tiger Kylesten

Supervisor : David Bergström
Examiner : Fredrik Heintz

External supervisors : Felix Härnström, Johan Sjöberg, Robert Wahlberg and Markus Wester

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Research on automated anomaly detection in complex systems by using log files has been on an upswing with the introduction of new deep-learning natural language processing methods. However, manually identifying and labelling anomalous logs is time-consuming, error-prone, and labor-intensive. This thesis instead uses an existing state-of-the-art method which learns from PU data as a baseline and evaluates three extensions to it. The first extension provides insight into the performance of the choice of word embeddings on the downstream task. The second extension applies a re-labelling strategy to reduce problems from pseudo-labelling. The final extension removes the need for pseudo-labelling by applying a state-of-the-art loss function from the field of PU learning. The findings show that FastText and GloVe embeddings are viable options, with FastText providing faster training times but mixed results in terms of performance. It is shown that several of the methods studied in this thesis suffer from sporadically poor performances on one of the datasets studied. Finally, it is shown that using modified risk functions from the field of PU learning provides new state-of-the-art performances on the datasets considered in this thesis.

Acknowledgments

We would like to thank our excellent supervisors at Sectra: Markus Wester, Johan Sjöberg, Felix Härnström and Robert Wahlberg for their guidance, support, and patience with us throughout the thesis. Further, we thank our supervisor David Bergström and examiner Fredrik Heintz for their encouragement and for being a source of inspiration during the thesis work. Without you this work would not have been possible.

Contents

Abstract iii

Acknowledgments iv

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Aim . . . 2
- 1.3 Research questions . . . 2
- 1.4 Delimitations . . . 3
- 1.5 Author's contribution . . . 3

2 Theory 4
- 2.1 Previous work . . . 4
- 2.2 Overview of pre-processing . . . 5
- 2.3 Word Embeddings . . . 5
- 2.4 Learning from positive and unlabelled data . . . 7
- 2.5 PLELog . . . 13
- 2.6 Attention . . . 15

3 Method 17
- 3.1 Baseline . . . 17
- 3.2 FastText (FT) . . . 18
- 3.3 Iterative Relabelling (IR) . . . 18
- 3.4 Alternate risk function (nnPU, Imb-nnPU) . . . 19
- 3.5 Experimental setup . . . 19

4 Results 24
- 4.1 Prior optimisation . . . 24
- 4.2 Varying known normals . . . 26
- 4.3 Lowering the number of known normals . . . 33
- 4.4 Introducing Anomalies . . . 34
- 4.5 Efficiency benchmark . . . 37
- 4.6 In relation to other works . . . 37

5 Discussion 39
- 5.1 Results . . . 39
- 5.2 Method . . . 41

5.3 The work in a wider context 43

6 Conclusion **45**
6.1 Research Questions 45
6.2 Future Work 46

Bibliography **47**

A Hardware **51**

B Varying known normals full tables **52**

C Threshold together with PLE **62**

D Hyperparameters **64**

List of Figures

3.1 Classification baseline 18
3.2 Training pipeline with re-labelling 19
3.3 Illustration of the six splits of the known normal sequences used in performance evaluation 21

4.1 Performance with varying priors 25
4.2 Lowering the number of known normals. The lines display the median value. Note the different scale on the x-axis for (c) and (d). 33

List of Tables

2.1 Notation of key PU learning concepts . . . 9
2.2 The dimensions for the parameter matrices used in the Multi-Head Attention. In the original paper $h = 8$, $d_{model} = 512$, and $d_k = d_v = d_{model}/h = 64$. . . . 16

3.1 Information about the datasets . . . 20
3.2 The number of anomalous sequences corresponding to 5% and 1% in HDFS and BGL. . . . 22

4.1 Optimal priors for *nnPU* and *Imb-nnPU*. . . . 25
4.2 Performance for the different models with varying known normals on BGL. . . . 27
4.2 Performance for the different models with varying known normals on BGL, continued. . . . 28
4.3 Performance for the different models with varying known normals on HDFS. . . . 29
4.3 Performance for the different models with varying known normals on HDFS, continued. . . . 30
4.4 Performance for the different models with varying known normals on SBDS. . . . 31
4.4 Performance for the different models with varying known normals on SBDS, continued. . . . 32
4.5 Introducing anomalies BGL with no threshold . . . 35
4.6 Introducing anomalies HDFS . . . 36
4.7 Time for pre-processing and vectorisation for each dataset in seconds. Includes training and/or loading models. . . . 37
4.8 Time taken for each model. Results over the dashed lines are on BGL and below on HDFS. . . . 37
4.9 Performance of the best method in this work together with other works in the field as reported in [41]. . . . 38

A.1 Hardware in computer A . . . 51
A.2 Hardware in computer B . . . 51
A.3 Hardware in computer C (Azure system Standard_NC6 [1]) . . . 51

B.1 Performance for the different models with varying known normals on BGL. . . . 53
B.1 Performance for the different models with varying known normals on BGL, continued. . . . 54
B.1 Performance for the different models with varying known normals on BGL, continued. . . . 55
B.2 Performance for the different models with varying known normals on HDFS. . . . 56
B.2 Performance for the different models with varying known normals on HDFS, continued. . . . 57
B.2 Performance for the different models with varying known normals on HDFS, continued. . . . 58
B.3 Performance for the different models with varying known normals on SBDS. . . . 59

B.3 Performance for the different models with varying known normals on SBDS, continued. 60
B.3 Performance for the different models with varying known normals on SBDS, continued. 61

C.1 Introducing anomalies BGL with threshold 63
C.2 Introducing anomalies HDFS with threshold 63

D.1 Hyperparameters 64

1 Introduction

The modern world increasingly relies on automated computer systems for everything from parking your car to diagnosing diseases in hospitals. With each passing year, the demands on these systems increase and with it the required complexity of the systems. These complex systems need to be available twenty-four hours a day, seven days a week. For some applications it is critical that the systems never fail when they are required. To cope with these extreme requirements, automated monitoring by use of system logs is used by virtually every automated system. These logs are used to provide invaluable information about the circumstances that caused the issue to the engineers. However, as the size and complexity of computer systems increase, so does the size of the system logs produced. The increase in log data makes it more and more difficult for humans to use the data contained in the log files to detect system errors. A real-world example of this is Alibaba Inc. whose cloud services today generate about 30-50 GB of logging data per *hour*, and the data velocity is expected to increase even further in the future. To cope with the extreme amount of data automated tools are being developed to assist in detecting anomalous events without requiring an engineer to manually sift through the data. Recent breakthroughs in the field utilising deep learning methods have enabled such services to achieve high precision, and automatic log anomaly detection models are currently in use at several large companies such as Microsoft and Huawei [16, 9]. These systems help the engineers detect errors and bugs in the systems early and automatically, allowing them to fix errors before the customer notices the degraded performance.

A common issue in automatic log anomaly detection systems by way of machine learning is that manually finding and labelling anomalies in log files is both tedious and error-prone. For this reason, most automatic systems have used unsupervised learning methods to produce anomaly detection models. However, producing log files from a system that works as expected, i.e., where we expect to find zero anomalies, is much easier! This makes the problem fit snugly in the field of positive-unlabeled learning (PU learning), a research area that has received a lot of attention recently. PU learning is a subfield of semi-supervised binary classification where access to a small set of labelled positive examples is assumed, together with a larger unlabelled set of mixed positive and negative examples. PU data arises frequently in real-world scenarios, two examples being in disease prediction where a set of patients have

the disease and a larger set may or may not have the disease or fraud detection where the bank has a small set of known fraudulent activity and a much larger set of unlabelled transactions. Recently, with the introduction of cost-sensitive empirical risk minimisation techniques, the performance of PU learning techniques have greatly improved. While learning from PU data in automated log anomaly detection has been tried previously [41, 39, 24], this is as far as we are aware the first work applying cost-sensitive empirical risk minimisation to the task.

1.1 Motivation

In this thesis, the task is to correctly identify system anomalies by studying the log files produced by the system. The scenario assumed in the thesis is that only PU data is available. That means that we have log files of systems that are known to be working as expected and a larger set of log files from systems that may or may not be working as expected. The field of PU learning has received extensive research attention recently and has been applied in a variety of fields from genomics to recommendation systems [35, 43]. This thesis adds to the application of PU learning methods to the field of log anomaly detection.

The industrial and research interest in the field of log anomaly detection has been skyrocketing recently [42, 41, 25, 7, 14] after Deeplog showed that it was possible to get effective anomaly detection using deep learning techniques. Using automated anomaly detection systems built on PU learning provides certain advantages over the traditional method of manually searching for errors. Firstly, the system can detect anomalies that the domain experts are unaware of, e.g. new anomalous behaviour introduced in a system update. Secondly, the model can be trained autonomously on generated test data which lowers the risk of accidentally mislabelling normal sequences as anomalous. Finally, the system can be built-in in addition to the existing pipelines, keeping the work already performed to detect system anomalies and reducing the future workload of the engineers responsible for system diagnostics.

1.2 Aim

The aim of this study is to evaluate the performance of different methods utilising PU data for automatic log parsing on known benchmark datasets and log datasets from real-world sites provided by Sectra. The focus is to evaluate the effectiveness of the models in a real-world setting. This implies certain restrictions on the pipeline, e.g. the memory consumption and processing time must be low enough to be non-prohibitive in a practical use case. Finally, this thesis aims to extend previously proposed state-of-the-art methods in automated log anomaly detection and investigate the performance difference produced.

1.3 Research questions

1. *How do the word embeddings impact the f1-score of the downstream task?*

   There are various methods to convert text data to numerical representations. While some techniques such as term-frequency inverse-document frequency (TF-IDF) are useful almost everywhere, the performance of other techniques differs between use cases.

2. *Which methods are appropriate for training anomaly detection models on normal and unlabelled log files, and what is their impact on the f1-score of the downstream task?*

   The field of PU learning contains different techniques for learning a classifier. These techniques can be split into four main branches, which will be investigated and applied to the task of anomaly detection.

3. *Is the system usable in a live system in terms of effectiveness and efficiency?*

   It is notoriously easier to get better performances with larger models, but larger models also incur costs in terms of memory and processing time. For a system to be deployed and used in live systems producing a large amount of log data, the system must be lightweight enough to be able to use in real-time, and it is crucial that the number of false alarms is kept low because of the high velocity of log data generated by the systems. For a system to be usable in a live system it must therefore strike a balance between accuracy and efficiency.

1.4 Delimitations

This thesis stops before root-cause analysis. A single system fault can produce a considerable number of anomalies, sometimes in the tens of thousands. Each of these anomalies will be reported individually, disregarding whether the root cause is shared or not. This necessarily means that the number of reported incidents from the model might be higher than the true number of system errors.

1.5 Author's contribution

This thesis has been co-authored by Tiger Kylesten and Henrik Ahlinder, and therefore this section aims to broadly explain how the work has been divided between the two authors. We stress that most of the work has been divided evenly and for implementation purposes, both authors have implemented at least some part of everything described in the thesis. Tiger Kylesten has taken the primary responsibility for the research necessary to answer research question one, while Henrik Ahlinder has had the primary responsibility for answering research question two. The implementation work has been shared with Tiger has researching and implementing DRAIN and the classifier for the baseline, while Henrik implemented the pre-processing and pseudolabelling process. Finally, with the implementation finished, Tiger took the primary responsibility for running the test and summarizing the results in the results section, while Henrik spent took the primary responsibility for writing the report.

2 Theory

This chapter aims to give a broad theoretical background on the theory and related work on automated log analysis. It begins with a short introduction of significant previous works in section 2.1 with the purpose of introducing the reader to the work done to reach the current state-of-the-art models. After covering the history, the general background theory of the problem formulation is discussed in section 2.2 followed by a detailed explanation of word embeddings and PU learning in section 2.3 and section 2.4 respectively. Section 2.5 gives an overview explanation of a state-of-the-art model, PLELog, that we have based our work on. Finally, this chapter is finished in section 2.6 where the concept of multi-head attention is described.

2.1 Previous work

The pioneering DeepLog introduced deep learning to the field of automated log analytics, producing state-of-the-art results using a Bi-LSTM network architecture. It has since then inspired new autonomous methods using the power of neural networks to model complex data to learn what a normal log pattern is, and thereby identify anomalous patterns. Several extensions of the basic ideas underlying DeepLog have been produced. Two of the most successful, PLELog [41] and LogRobust [42], improve the robustness by taking the semantic meaning of log messages into consideration and increase the performance by utilising attention in the classifier.

A crucial weakness of DeepLog and derivations of it is that log data is incredibly laborious and difficult to label, and the models must therefore resort to unsupervised training methods. DeepLog is trained to predict the next log message and reports anomalies based on the estimated likelihood of the actual next log message. PLELog [41] uses another approach and attacks this problem by creating pseudo-labels for the data points based on the assumption that we have access to a small sample of known normal log sequences. The performance increase for PLELog compared to DeepLog is significant, but it still performs worse than the fully supervised method LogRobust [42].

2.2 Overview of pre-processing

While different methods are used to train neural log anomaly detection models, there are some parts which are shared between most of them. This section aims to provide background information about these shared steps. The first subsection discusses how the log file is processed into sequences and the section is subsequently finished with a subsection discussing how the raw log messages are parsed into structured data.

Sequencing the log files

There is no consensus in the current literature on how to split a log file into sequences, and different papers use different methods. Unfortunately, this difference is significant for the performance of the models which makes it difficult to make a fair comparison between algorithms by the results presented in published papers. There are three primary methods used: dividing the log file into a group of sequences using a fixed window, a sliding window or a session window [17]. While the first two are self-explanatory, session window requires an explanation. A session window split is based on finding an identifier which can be used to divide the log file into windows, usually by capturing a process execution in each window. Therefore, splitting the dataset into session windows gives a better performance, but it requires that the log file contains identifiers and the domain expertise to recognise them [17].

Further, there are two commonly used methods to split the sequences into train, test, and validation data. The first is the usual approach in machine learning, used for instance by [42, 12], which is to randomly shuffle the data and separate it into three sets. However, [41] argues that in this scenario, a more natural separation is to choose the chronologically earliest windows as training data, as this separation method is a better approximation of the real-world use-case where the model will be used to predict anomalies in log files produced chronologically after the model training. It is worth noting that splitting the data chronologically might induce a lower performance since it increases the risk of introducing previously unseen data in the test and validation sets.

Log parsing

Finally, after selecting a dataset and separating it into sequences the final step is to convert the raw log messages to structured data.

Since the model is effectively trying to separate normal and abnormal execution patterns, it makes intuitive sense to try to identify from which part of the program a certain log message originated. This is done by utilising the fact that log messages almost always consist of a static template with a few variable parts. Therefore, if we can find templates which accurately match the log messages using wildcard replacements, it is likely that these will match the original static template in the executing code. Various tools exist that parse logs efficiently and effectively. A recent study of different log parsing algorithms was done by [44], where the advantages and disadvantages of each method were thoroughly discussed. The most commonly used tool in log anomaly detection is called Drain3 and was determined to be one of the best-performing log parsers by the survey. A more detailed explanation of Drain3 is given in section 2.5

2.3 Word Embeddings

A common initial problem when processing text is that computers only understand numerical data, and has no natural way to process text data. A naïve way to solve this problem is to one-hot encode each word in the vocabulary. However, this method is neither space efficient with very large vocabularies, nor is it a good way to represent the data. A better

representation is word embeddings. Word embeddings are a popular technique used in natural language processing that involves representing words as vectors in a high-dimensional space. The basic idea behind word embeddings is that words that are similar in meaning or context should be represented by similar vectors in this space. In other words, words that occur in similar contexts should have similar embeddings. For example, the words ”cat” and ”dog” are semantically similar because they are both animals, and they are syntactically similar because they are both nouns. Word embeddings attempt to capture these similarities by mapping each word to a vector in a high-dimensional space, such that words with similar meanings and contexts are mapped to similar vectors.

Word embeddings are usually learned by training a neural network on a large corpus of text data. The neural network takes in a sequence of words as input and outputs a set of vectors that represent the words in the sequence. One popular neural network architecture used for this task is Word2Vec [26], which trains a model to predict the context of a given word in a sentence. Another popular algorithm is GloVe [29], which is based on matrix factorisation and aims to capture the co-occurrence statistics of words in a corpus.

Once learned, word embeddings have been proven suitable for use in a wide range of natural language processing tasks, leveraging their ability to represent semantic relationships between words in a space-efficient way. Two different popular methods of extracting word embeddings that are used in this thesis, GloVe and FastText, are described in separate subsections below.

GloVe

GloVe, or Global Vectors for Word Representation, is a popular algorithm for generating word embeddings that have been widely used in natural language processing (NLP). The method is based on the idea of constructing a co-occurrence matrix that counts the number of times each word appears in the context of every other word in a given corpus. The matrix is then factorised using singular value decomposition to obtain low-dimensional vector representations for each word. The resulting vectors capture the semantic and syntactic relationships between words in the corpus and can be used in a variety of NLP tasks [29].

One of the main advantages of GloVe over other methods for generating word embeddings is its ability to capture complex relationships between words that are not easily captured by other methods. For example, it can capture the difference in meaning between words with different polarities, such as “good” and “bad”, or words with different degrees of intensity, such as “happy” and “ecstatic”. Another advantage of GloVe is its efficiency in training on very large corpora, thanks to its use of matrix factorisation, which is a computationally efficient method for generating embeddings. Finally, GloVe has been used to generate pre-trained embeddings for a variety of languages, which can be used directly without the need for additional training. These pre-trained embeddings have been part of the architecture of many state-of-the-art algorithms in natural language processing.

FastText

FastText is an algorithm developed by the Facebook research team in 2016 [6]. The intuition behind the algorithm is that by extending the continuous skip-gram algorithm [27] to analyse tokens on a sub-word level more morphological information can be retained and therefore create more representative word embeddings. In contrast to the previously discussed GloVe, FastText trains word embeddings through the use of a feed-forward neural network. To understand FastText we first need to understand the skip-gram algorithm with negative sampling was introduced by [27]. The intuition behind the algorithm is that tokens which occur in the same context should have similar embeddings and tokens that do not should

have dissimilar embeddings. In practice, the algorithm takes a target word w_t, mines relevant context words $\bar{w}_c$ and samples a set of negative context words $\mathcal{N}_{t,c}$ from the dictionary. Thereafter, for a chosen context position c and using the binary logistic loss, the following negative-log-likelihood is obtained:

$$log\left(1 + e^{-s(w_t, w_c)}\right) + \sum_{n \in \mathcal{N}_{t,c}} log\left(1 + e^{s(w_t, n)}\right)$$

where $s(w_t, w_c) = \mathbf{u}_{w_t}\mathbf{v}_{w_c}$ is the score function based on the dot-product similarity of the word embeddings for the target word w_t and context word w_c. In other words, this objective function rewards word embeddings that put context and target words close to each other, while ensuring that negative samples are far away from the target word.

FastText improves on the continuous skip-gram algorithm by modifying the score function s to incorporate the internal structure of words by representing each word w as a bag of character n-grams on top of the word w. Additionally, the characters '<' and '>' are added to indicate the start and end of a word. For example, using the word *where* and $n = 3$, the extracted representation will be:

<wh, whe, her, ere, re>, <where>

Note that the addition of the start and end tokens separates the trigram *her* from the word *<her>*. Finally, the embedding for a word w is calculated by the sum of the vector representations of its n-grams. Therefore, the modified scoring function s becomes:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} z_g^T w_c.$$

where z_g is a vector representation of n-gram g and w_c is a vector representation of the context word. In practice, all n-grams for $n \in 3, 4, 5, 6$ are used during training since this configuration has been shown to provide the best compromise between performance and training time.

The main advantage of the slightly more complex training algorithm of FastText is that the incorporation of character n-gram allows learning reliable representations for rare words by sharing representations across words. As an example, *wrongly* rarely occurs in texts, but by understanding the more common word *wrong* together with the frequently used suffix *-ly* we can infer an understanding of the word *wrongly*. Additionally, using character n-grams helps alleviate the issue with out-of-vocabulary words since the meaning of previously unseen words frequently can be inferred by the subwords it is made of. A word such as *unfriendly* might never occur in the training data, but as in the example above it can still be understood by its subparts: *un*, *friend* and *ly*. Non-subword algorithms have no tools to handle such scenarios and therefore cannot interpret the semantic meaning of out-of-vocabulary words.

2.4 Learning from positive and unlabelled data

PU learning, or Positive-Unlabelled learning, is a type of semi-supervised binary classification where the labelled dataset consists entirely of one class. This scenario arises frequently in many real-world scenarios where obtaining negative examples is difficult or expensive. For example, in medical diagnosis there may be cases where a patient has a rare disease that is difficult to diagnose. In such cases, the patient with the rare disease is the positive example, while the rest of the patients are considered unlabelled as the absence of proof is not the same as the patients not having the disease. The task of anomaly detection on log files is a special case of PU learning where it is relatively easy to produce normal log sequences using test systems, but laborious and error-prone to find and label anomalous log-lines.

It has been shown that classifiers can be trained on PU data with different assumptions and solved using different methods. At its heart however, the goal of PU learning is to train a classifier that can distinguish between two classes based on the attributes of an example. This section therefore begins by reviewing the basics of binary classifications before the setting of PU learning is described which is followed by a description of the two different scenarios encountered in PU learning, the *Single Training Set* and *Case Control* scenarios. Following this background, various methods and their respective assumptions are discussed. Finally, this section is finished by explaining the method used in this thesis (*empirical risk minimisation*) and how to apply it in practice to imbalanced data. All content in this section, if nothing else is stated, can be assumed to be from [2] which provides an exhaustive overview of the topic.

Binary Classification basics

Binary classification is the task of training a classifier to distinguish between examples from two different classes. By convention these are called ”positive” and ”negative” classes. Note that in this report the positive class represents normal sequences and negative anomalous sequences; the opposite of what the classes usually refer to. This is for the purpose of keeping the terminology clear when discussing PU learning theory, since the labelled class consists of normal sequences. Each training example is a tuple (x, y) where x is the vector of attribute values and y is the class. An example is positive if $y = 1$ and negative if $y = 0$. Traditional learning algorithms work in a supervised setting, where the training data is assumed to be fully labelled. That is, the class value for each training example is observed. To enable training a classifier, the training data is assumed to be an independent and identically distributed (i.i.d.) sample of the real distribution:

$$\mathbf{x} \sim f(x) \sim \pi f^+(x) + (1 - \pi) f^-(x) \tag{2.1}$$

with the class prior $\pi = P(y = 1)$ and probability density functions of the true distribution f and the positive and negative examples f^+ and f^- respectively.

PU Learning

A PU dataset is usually represented as a set of triplets, (x, y, s) where x is a vector of attributes, y is the class and s is a binary indicator variable of whether the example was selected to be labelled. The class y in this setting is never observed, but information about it can be inferred by the value of s. If $s = 1$ for an example, then it belongs to the positive class: $\implies P(y = 1|s = 1) = 1$. When the example is unlabelled ($s = 0$) it can belong to either class. We use the same notation as [2], which is the most common notation to use. Table 2.1 gives an overview of the notation.

Labelled positive examples are selected from the complete set of positive examples according to a probabilistic labelling mechanism where each positive example x has the probability $e(x) = P(s = 1|y = 1, x)$ of being selected, called the *propensity score* [3]. Therefore, the labelled distribution is a biased version of the positive distribution:

$$f_l(x) = \frac{e(x)}{c} f^+(x) \tag{2.2}$$

where $f_l(x)$ and $f^+(x)$ are the probability density functions of the labelled and positive distributions respectively and c the label frequency, i.e. the fraction of positive examples that are labelled $c = P(s = 1|y = 1) = \mathbb{E}_x[e(x)]$.

The Single-Training-Set and Case-Control Scenarios

The positive and unlabelled examples in PU data can originate from two scenarios. Either they come from a single training set, or they come from two independently drawn datasets, one with all positive examples and one with all unlabelled examples. These scenarios are called the single-training-set scenario and the case-control scenario respectively.

The single-training-set scenario assumes that the positive and unlabelled examples come from the same dataset and that the dataset is an i.i.d. sample of the real distribution. A fraction c of the positive examples is selected to be labelled given their propensity scores. Therefore, the dataset has the following distribution:

$$\begin{aligned} \mathbf{x} &\sim f(x) \\ &\sim \pi f_+(x) + (1-\pi) f_-(x) \\ &\sim \pi c f_l(x) + (1-\pi) f_u(x). \end{aligned}$$

The case-control scenario instead assumes that the positive and unlabelled examples come from two independent datasets and that only the unlabelled dataset is drawn from the real distribution. The distribution then becomes:

$$\begin{aligned} \mathbf{x}\,|\mathbf{s}=0 &\sim f_u(x) \\ &\sim f(x) \\ &\sim \pi f_+(x) + (1-\pi) f_-(x). \end{aligned}$$

The single training set scenario has received substantially more attention in the literature, but in both scenarios the classifier has access to a set of positive examples drawn i.i.d from the true distribution and a set of examples that are drawn from the positive distribution according to the label mechanism that is defined by the propensity score $e(x)$. Therefore, most methods can handle both scenarios. Only methods that can handle both scenarios are used in this thesis, and therefore no further consideration of this problem will be made.

Methods for PU learning

The various methods of PU learning can be divided into four different categories:

1. *Two-Step Techniques* attempt to determine reliable negative samples (and in some cases further positive observations) in the initial step, and then in a second step uses the la-

Table 2.1: Notation of key PU learning concepts

Symbol	Description
x	A vector of attributes for a single example.
$\mathbf{x}$	A set of attribute vectors.
y	Class label for a single example
s	Binary indicator for a single example to be labelled
π	Class prior: $P(y=1)$
c	Label frequency : $c = P(s=1\|y=1)$
$e(x)$	Propensity score function: $e(x) = P(s=1\|y=1,x)$
$f(x)$	Probability density function of the true distribution
$f^+(x)$	Probability density function of positive examples
$f^-(x)$	Probability density function of negative examples
f_l	Probability density function of labelled examples
f_u	Probability density function of unlabelled examples
$\hat{\bullet}$	An estimate for $\bullet$

bels to apply traditional semi-supervised learning approaches to the generated positive, negative, and unlabelled observations.

2. *Biased Learning* approaches consider PU learning as a binary classification problem with the unlabelled samples as noisy observations of the negative class. For this, normal binary classifiers are modified such that misclassifications of positives are penalised harder than those of negatives or hyperparameters of learners are tuned to optimise appropriate PU metrics in the validation dataset.

3. *Class Prior Incorporation* uses direct knowledge about the class prior π to generate a good classifier. A distinction can be made between pre-processing and post-processing. In pre-processing a modified dataset is created by rebalancing, incorporation of label probabilities or empirical risk-minimisation methods, and directly integrated into the learning process. In post-processing the decision function or the predicted probabilities are adjusted after training.

4. Finally, *Other Methods* summarise approaches which cannot be assigned to the above categories, e.g. modelling of the densities of the two classes by generative adversarial networks adapted to PU data.

The baseline method used by PLELog [41] to assign labels through density-based clustering is an example of a two-step approach. Apart from that, this thesis will delve further into techniques based on class prior incorporation. In particular, cost-sensitive empirical risk-minimisation methods that can be applied to unstructured text data are explored.

Assumptions

Methods using class prior incorporation require that the class prior π is known. This is however not always the case, and estimating it from the data requires assumptions about the data and labelling mechanism to be made. For this thesis two assumptions are central:

1. Separability This assumption states that the two classes of interest are naturally separated. Therefore, a classifier exists that can perfectly distinguish positive from negative examples. This idea is exploited by two-step approaches that incrementally provide labels to the unlabelled based on the certainty of the model since this assumption tells us that a good classifier should classify all labelled examples and simultaneously as few unlabelled examples as possible as positives.

2. Smoothness This assumption states that examples that are close to each other are more likely to have the same label. This idea allows the extraction of reliable negatives as examples that are far away from all the negative examples, using a suitable distance metric.

The positive distribution shown in equation 2.2 depends on the propensity score $e(x)$ which differs depending on the underlying assumptions made of the data. Under the assumption of a *probabilistic gap*, given as $\Delta p(\mathbf{x}) = p(y = 1|x) - p(y = 0|x)$, $e(x)$ is calculated by an estimation of the probabilistic gap. This means that the likelihood of labelling a positive sample is proportional to how similar it is to the negative samples.

Under the *Selected At Random (SAR)* assumption the propensity score depends entirely on $\mathbf{x}$ but there is no probabilistic gap. Finally, the most commonly researched assumption is the *Selected Completely At Random (SCAR)*, where the propensity score is independent of $\mathbf{x}$ thus

$$e(x) = p(s = 1|y = 1, x) = p(s = 1|y = 1) = c, \tag{2.3}$$

i.e. $e(x)$ becomes the label frequency.

Class Prior

A central aspect of PU learning is the class prior $\pi = p(y = 1)$. The class prior has a direct relationship with the label frequency:

$$\begin{array}{lcll} c & = & \frac{P(s=1)}{\pi} & \textit{Single training set scenario} \\ c & = & \frac{P(s=1)}{\pi(1-P(s=1))+P(s=1)} & \textit{Case control.} \end{array}$$

The prior can be used in several ways to train a classifier, and it is frequently used together with the SCAR assumption. In some applications, neither π nor c are known and must be estimated from the data. Suitable methods for this include but are not limited to partial matching [30], receiver operating characteristics [5] or decision tree induction [33]. In this thesis, following the arguments of [11], the prior π is instead treated as a hyperparameter and a thorough search for an optimal value for it is done for each dataset.

Empirical Risk Minimisation

One of the popular methods for learning from PU data is class prior incorporation as described in 2.4. This area has received a lot of research attention in recent years, with a particular focus on methods attempting to select a classifier by minimising a suitable risk function. Usually, instead of optimising a classifier $f(x) : \mathbb{R}^d \to \{0, 1\}$, a function $g(x) : \mathbb{R}^d \to [0, 1]$ which is designed to approximate $P(y = 1|x)$ is optimised. In other words, we seek a suitable risk function $R(g)$ to use for selecting $g(x)$ as:

$$g(x) = \arg\min_{g \in \mathcal{G}} \hat{R}(g)$$

Using the fact that $g(x) = \hat{P}(y = 1|x)$ a suitable classifier to use is

$$f(x) = \begin{cases} y = 1, & g(x) \geqslant 0.5 \\ y = 0, & \text{otherwise} \end{cases}$$

In traditional supervised classification the misclassification risk $R(g) = \mathbb{E}_{(X,Y)\sim p(x,y)}[\ell(g(x), y)]$ is commonly used to find $g(x)$. In the binary case it can be rewritten as:

$$R_{PN}(g) = \pi R_P^+(g) + (1 - \pi)R_N^-(g), \tag{2.4}$$

where $R_P^+(g) = \mathbb{E}_{P(x|y=1)}[\ell(g(x), 1))]$ and $R_N^-(g) = \mathbb{E}_{P(x|y=0)}[\ell(g(x), 0))]$ is the misclassification risk for positive and negative samples, or the *false negative* and *false positive* risk, using zero-one loss:

$$\ell(g(x), y) = \begin{cases} 0, & y = g(x) \\ 1, & y \neq g(x) \end{cases}$$

Since we do not have access to the true negative labels equation 2.4 is unusable. However, since $f_u(x) = \pi f_+(x) + (1 - \pi)f_-(x)$ in both scenarios listed in section 2.4, [13] shows that $R_N^-(g)$ can be approximated indirectly by $(1 - \pi)R_N^-(g) = R_U^-(g) - \pi R_P^-(g)$, i.e. treating all

unlabelled samples as negatives and removing the bias produced by positive samples in the unlabelled set. This leads to the formulation of a pure PU risk function:

$$R_{pu} = \pi R_P^+(g) + R_U^-(g) - \pi R_P^-(g) \tag{2.5}$$

In practice, it is found that due to the strong fit ability of neural networks the empirical estimate of the second term of equation 2.5 can go much lower than zero. However, theoretically, this term is used to estimate $R_N^-(g) = (1-\pi)\,\mathbb{E}_{P(x|y=-1)}[\ell(g(x),0)]$, which should always be non-negative. Therefore, it was proposed in [21] to instead optimise the non-negative risk displayed in 2.6. This risk equation is called the *nnPU* (non-negative PU) risk and is a state-of-the-art solution in the field of PU learning.

$$\hat{R}_{nnpu} = \pi_P \hat{R}_P^+(g) + \max(0, (\hat{R}_U^-(g) - \pi_P \hat{R}_P^-(g))). \tag{2.6}$$

Surrogate loss

While the zero-one loss formulation is useful in theory, in practice it is difficult to optimise. It is therefore suitable to replace it when training with a surrogate loss function [13]. A common replacement is the sigmoid loss

$$\ell_{sig}(g(x), s) = \frac{1}{1 - \exp(2(s-0.5)g(x))}.$$

Addressing Imbalanced Data

Previously the nnpu-risk was derived and is presented in equation 2.6, which has shown excellent results in various PU learning scenarios across different domains. However, it does not account for class imbalance, which is a common issue in anomaly detection problems. Especially problematic is this issue in PU learning where oversampling the minority negative class or downsampling the majority positive class is not feasible because only labelled positives are known and there may be more positives in the unlabelled set U. To address this, [36] developed a reweighting strategy for imbalanced PU data that simulates oversampling the minority class and demonstrates its superiority to other approaches on imbalanced data. While their paper defines the positive class as the minority, in our case it is the negative class. This does not impact the method's validity, and therefore we keep the same notation as previously in the thesis. The new risk function, called *ImbalancednnPU*, is represented by Equation 2.7, which includes an additional weight term $\frac{1-\pi'}{1-\pi}$ to account for class imbalance

$$R_{\text{ImbalancednnPU}}(P,U) = \pi' R_P^+(g) + \max\left(0, \frac{1-\pi'}{1-\pi}\left(R_U^-(g) - \pi R_P^-(g)\right)\right) \tag{2.7}$$

where $\pi' = P_{balanced}(y=1)$ represents the proportion of the positive samples in the distribution of the simulated dataset, which the author recommends set to 0.5. In practice, we need to optimise an empirical estimation of $R_{imbalancednnPU}$. For the sake of readability, we first denote the empirical estimate of the *false positive* risk $\hat{R}_N$ followed by the empirical estimate $\hat{R}_{ImbalancednnPU}$ as a function of the estimate R_N

$$\begin{aligned} \hat{R}_N^-(P,U) &= \frac{1-\pi'}{1-\pi}\left(\frac{1}{n_u}\sum_{x_i\in U}\ell(g(x_i),-1) - \frac{\pi}{n_p}\sum_{x_i\in P}\ell(g(x_i),-1)\right) \\ \hat{R}_{ImbalancednnPU}(P,U) &= \frac{\pi'}{n_p}\sum_{x_i\in P}\ell(g(x_i),1) + \max(0,\hat{R}_N^-(P,U)), \end{aligned} \tag{2.8}$$

where n_p and n_u are the number of positive and unlabelled samples respectively, and $\ell(\bullet,\bullet)$ is a surrogate loss function.

Flooding

Flooding [19] is a technique to reduce overfitting to the training data, allowing the model to train for a longer time and has empirically been shown to increase the model performance for stochastic optimisers. By setting a sufficiently large minimum training loss τ, the so-called *flood level*, the authors argue that a zero training *error* does not require a zero training *loss*. The intuition is that the model should continue to learn without resorting to memorising the training examples, which is what happens when the training loss becomes near zero. In other words, if the original learning objective is J, the modified learning objective $\tilde{J}$ with flooding is:

$$\tilde{J}(\theta) = |J(\theta) - \tau| + \tau,$$

where $\tau > 0$ is the flood level as specified by the user and θ are the model parameters.

In practice, making the training loss float around τ is achieved by alternating normal gradient descent with gradient ascent steps. In other words, if an optimisation step would result in a lower training loss than the threshold τ a gradient *ascent* is performed instead of the normal gradient descent. As the training loss bobs above and below τ the model will do a "random walk" and is then expected to drift into an area with a flat loss landscape that leads to better generalisation [19].

Flooding is frequently used in tandem with cost-sensitive risk-minimisation methods by tracking the value of $\hat{R}_N^-(P,U)$ [36, 8, 21]. If it is negative the model has overfitted to the training data and a gradient ascent step is performed instead of the usual gradient descent step.

2.5 PLELog

PLELog is one of the best-performing semi-supervised models currently on the task of log anomaly detection. The primary innovation enhancing its performance is the utilisation of known normal sequences, which are used to produce probabilistic labels that a standard binary classifier subsequently learns from. This section aims to briefly explain PLElog since many of the same principles will be used by the classifier used in this thesis. If nothing else is stated, all contents of this chapter can be assumed to be from [41] which introduced PLELog.

Sequencing the log files

In PLELog the log files are sequenced into session windows and subsequently split into train, validation, and test data using a chronological split. For more information about sequencing the log files, see section 2.2.

Log parsing with Drain3

PLELog uses the open-source tool Drain3 to parse raw log messages into structured data. Drain3 is a two-step approach, where the first step is to mask certain parts of the log messages based on hand-crafted rules, e.g. masking ip-adresses with the token *[IP]* and the second step is to group the modified log messages into clusters based on the number of identical words in the log message. Finally, a template is extracted from each cluster. This template is built to match all log messages belonging to the cluster and is produced by replacing certain words in log messages with a wildcard token [15].

After replacing all raw log messages with their respective templates, the final step before vectorisation is to tokenise the templates produced by Drain3. The tokenisation is done by first separating concatenated identifiers in camel case and snake case (e.g. BusFactory→bus factory, bus_factory→bus factory). Splitting identifiers is important since many programming languages do not allow whitespace separation of words, and therefore programmers resort to camel case or snake case separation of concatenated words. Finally, each whitespace-delimited word in the processed templates is treated as a token.

Vectorisation

After parsing the log, the next step is to transform the templates into a vector representation. The representation should have a low enough dimension to be easy to work with, but should still clearly represent the message. The goal is to make similar templates have close representations and dissimilar templates far away from each other in the representation space.

To vectorise the templates two commonly used methods in NLP are used in tandem: TF-IDF vectorisation and word2vec [29]. The word embeddings used were GloVe trained on six billion tokens from Wikipedia. To produce an embedding for a token, the word embedding is multiplied by the L1-normalised term frequency and inverse document frequency for the token. If the token does not exist in the GloVe embeddings it is ignored. Finally, the embedding for the template is calculated as the sum of each token embedding. The mathematical formula for this process is displayed below:

$$TE = \frac{1}{N} \sum_{token} \text{TF}_{\text{token}} \cdot \text{IDF}_{\text{token}} \cdot \text{Embedding}_{\text{token}}$$

where TE is the template embedding, TF_{token} is the token term frequency, N is the number of tokens in the template, $\text{IDF}_{\text{token}}$ is the inverse document frequency of the token, and $\text{embedding}_{\text{token}}$ is the word2vec embedding for the token.

Creating Probabilistic Labels

The primary innovation of PLELog over previous semi-supervised approaches is using an assumption of a set of known normal sequences to label the remaining dataset. After producing the labels the problem is therefore changed into a standard binary classification problem. The labels are predicted by clustering all the sequences in the training dataset, after which the assumption is made that sequences belonging to a cluster containing at least one known normal sequence are likely to also be normal. Of course, this assumption might be incorrect, especially for sequences on the edge of a cluster. Therefore, the certainty of each label is reduced by the outlier score (as calculated by GLOSH). The full span of the labels is therefore $[0, 1]$ instead of one-hot labels.

To facilitate the clustering each sequence must first be reduced to a single point. In PLELog this is accomplished by calculating the sequence embedding as the sum of each log event

embedding in the sequence. After this, the sequences are clustered using the algorithm HDBSCAN [23]. Since HDBSCAN (and clustering algorithms in general) deal poorly with high-dimensional data the dimensions of each sequence embedding is reduced to 50 using FastICA before clustering.

Classifier

The classifier consists of a multi-layer GRU (Gated Recurrent Unit) with an attention layer attached to the hidden states of the GRU. These attended hidden states are then summed along the sequences and projected down to scores for the classes using a linear layer [41]. For each element x_t in the input sequence $S = \{x_1, \ldots, x_n\}$ the first layer in the GRU calculates the reset gate r_t, the update gate z_t, the new gate n_t, and the hidden state at time t, h_t accordingly

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \tag{2.9}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \tag{2.10}$$

$$n_t = \tanh\left(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h(t-1) + b_{hn})\right) \tag{2.11}$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \tag{2.12}$$

where σ is the sigmoid function, $\odot$ is the Hadamard product and $h_0 = 0$. The second layer calculates the same equations, but with the input $x_t^{(2)} = h_t^{(1)}$, where $x^{(2)}$ denotes the input to layer 2, and $h_t^{(1)}$ denotes the hidden state for layer 1. W denotes learnable network parameters and b denotes learnable biases. [31]

To cope with vanishing gradients as a result of the long dependency issues associated with GRUs, an attention-based mask strategy is used in the same way as LogRobust [42].

2.6 Attention

Many state-of-the-art architectures for natural language processing, and more specifically, machine translation, have been using RNNs combined with an attention layer to put focus on certain parts of sequence data, and to cope with the vanishing gradients of RNNs. Even though GRUs helps to solve the problem of vanishing gradients, it isn't perfect [34], and an attention-layer together with a GRU has proven to give better results for longer sequences in machine translation [1].

LogRobust [42] uses a bidirectional LSTM together with an attention layer which takes the hidden states h_t from the LSTM and computes an attention weight $\alpha_t = \tanh(W_t^\alpha \cdot h_t)$, where W_t^α is the weight of the attention layer at timestep t. The attended hidden states $h_t^\alpha = \alpha_t \cdot h_t$ are then summed together.

Multi-Head attention

Recent state-of-the-art architectures for language modelling, e.g. BERT [10] GPT [32], have abandoned RNNs in favour of attention-only architectures called transformers which Vaswani et al. [37] introduced in 2017. The attention mechanism introduced is called Multi-Head Attention and takes as input a query Q, a key K and a value V.

Each head consists of a "Scaled Dot-Product Attention" which takes as input queries q, keys k – both with dimension d_k – and values v with dimension d_v. The output is computed as

$$\text{Attention}(q, k, v) = \text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)v$$

and the Multi-Head Attention is computed as

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

and W^Q, W^K, W^V and W^O are parameter matrices with dimensions seen in Table 2.2.

Table 2.2: The dimensions for the parameter matrices used in the Multi-Head Attention. In the original paper $h = 8$, $d_{model} = 512$, and $d_k = d_v = d_{model}/h = 64$.

Parameter matrix	Dimension
W_i^Q	$d_{model} \times d_k$
W_i^K	$d_{model} \times d_k$
W_i^V	$d_{model} \times d_v$
W^O	$h \cdot d_v \times d_{model}$

3 Method

This chapter presents the method used to answer the research questions posed in the introduction. The first section describes the fundamental classification pipeline, which is also used to produce a baseline performance. The baseline borrows almost all of its contents from PLELog [41]. The following sections each describe a different modification to the baseline. The first modification involves changes to the vectorisation step and is described in section 3.2. Following it, section 3.3 discusses incorporating a feedback loop to the probabilistic label estimation and section 3.4 introduces an alternative objective function and training procedure for the classifier. This chapter ends with section 3.5 where the experimental setup for the evaluation of the models is described.

3.1 Baseline

To get a baseline performance on the task, a modified version of PLELog was used. PLELog in its original form is described in section 2.5, but the baseline used in this paper has three minor modifications from the open-source implementation:

1. The CamelCase tokenisation is done prior to calculating template embeddings. This means that eventually separated identifiers by CamelCase will have equal importance in the template embedding vector.

   There is no reason to consider distinct parts of run-together words as words of lesser importance. Further, the tokenisation algorithm becomes easier to implement in practice when treated as separate words.

2. The word embedding for the special token *[UNK]* was used instead of purely zeros as word embedding for out-of-vocabulary tokens.

   Using zeros as the word embedding for out-of-vocabulary tokens is essentially equivalent to ignoring them. It is better practice to use the embedding for unknown or low-frequency words as the word embedding .

3. A different type of attention than the one implemented in PLELog was used. The attention used is a variant of the Multi-head attention described in 2.6.

The attention used by PLELog [41] uses a weight matrix for each time step t, and for varying length of the sequences this is innefficient since there will be many zero-multiplications due to the padding, and it enforces a max length for the sequences, which is unwanted.

The full pipeline for the baseline is visualised in Figure 3.1.

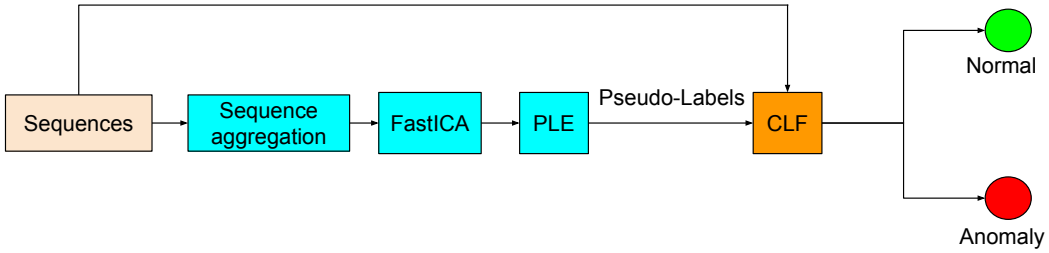


Figure 3.1: Classification baseline

3.2 FastText (FT)

This modification introduces a new method of calculating template embeddings. In the baseline implementation, GloVe is used to extract embeddings for each token. In this modification, new word embeddings were produced by training a FastText-model on the simple-wiki dataset and available log files. The reason for this is two-fold: the first is to include the specific terminology used in log files, avoiding the large number of out-of-vocabulary tokens encountered when using GLOVE, and the second is to utilise the character-based continuous *n*-gram model to help understand the shortened words frequently used in programming. More information about word embeddings can be found in section 2.3.

The FastText model was trained using unsupervised training from the open-source API [1] to produce word embeddings of size 50. The training data used to train the FastText model was the training data from the log file together with all words in lowercase from the pre-tokenised *wikitext-103* [2].

3.3 Iterative Relabelling (IR)

In the baseline, a label for each sequence in the dataset is calculated once at the beginning of the process. However, there is no guarantee that the first representation produced using FastText / GloVe is the best representation for producing the pseudo-labels. In fact, the objective of the RNN (as described in section 2.5) is to produce representations where anomalies are distinguishable from normal sequences. For that reason, it makes sense to attempt to re-label the data points using the classifier's representation after training on the initial labels for a few epochs.

In practice, this was done by the addition of a middle dense layer which takes the output from the RNN and projects it to 30 dimensions, which is followed by another dense layer which is used for the classification. The middle layer projection is used by the clustering algorithm when re-labelling the data points. The new network architecture can be seen in Figure 3.2. Each data point in the U was re-labelled using the same labelling strategy described in 2.5 every other epoch using the mid-layer projection as the sequence representation.

[1] https://fasttext.cc/

[2] https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset

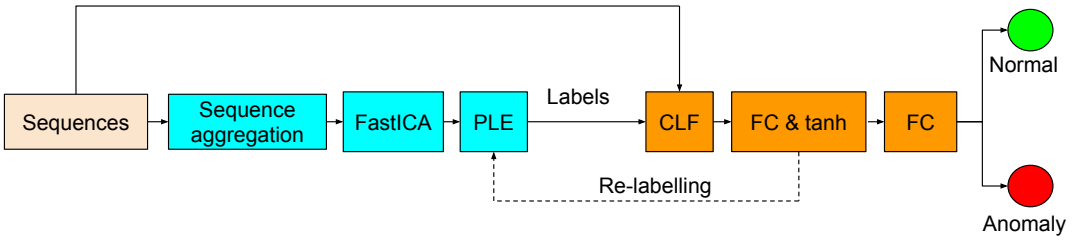


Figure 3.2: Training pipeline with re-labelling

3.4 Alternate risk function (nnPU, Imb-nnPU)

As described in section 2.4 the pseudo-labelling step of PLELog can be replaced if an alternate risk function is used which is purpose-built for learning from PU data. Two modified risk functions were used in this thesis: $\hat{R}_{nnPU}$ [21] and $\hat{R}_{ImbalancednnPU}$ [36], and they are referred to as *nnPU* and *Imb-nnPU* respectively. Section 2.4 presents the risk functions. The optimisation method used for both risk functions is displayed in algorithm 1. The step described on line 6 is normal gradient descent, while line 8 describes a commonly used overfitting measure. It states that, theoretically, $\hat{R}_N$ should never become negative but due to the strong fit ability of deep neural networks the estimate might become negative anyway. If it is, it is an indication of the classifier overfitting to the data, and therefore a negative gradient ascent is used instead of normal descent to avoid overfitting according to the flooding principle. See section 2.4 for details.

Algorithm 1 ImbalancednnPU

Input: Training data P and U
Parameter: class priors π and π'
Output: classifier $\hat{g}(x;\theta)$

1: let $\mathcal{A}$ be an SGD-like optimiser such as Adam [20]
2: **repeat**
3: Shuffle P and U into b mini-batches, each is represented as P_i and U_i respectively
4: **for all** batches P_i, U_i **do**
5: **if** $\hat{R}_N(P_i, U_i) \leqslant 0$ **then**
6: Update θ using $\mathcal{A}$ with the gradient $\Delta_\theta \hat{R}_{nnBalancePU}(P_i, U_i)$
7: **else**
8: Update θ using $\mathcal{A}$ with the gradient $-\Delta_\theta \hat{R}_N(P_i, U_i$
9: **end if**
10: **end for**
11: **until** Tired

Finally, the predictions $\hat{y}$ were produced by the decision rule

$$\begin{cases} \hat{y} = 1, & g(x) \geqslant 0.5 \quad \#Normal \\ \hat{y} = 0 & otherwise \quad \#Anomaly \end{cases}$$

3.5 Experimental setup

The experiments were run using a computer on Azure. The hardware specifications for the computer can be found in appendix A. The hyperparameters used for each experiment can be found in Appendix D or in the config files of our GitHub repository: [3]. Since the number of

[3] https://github.com/tigmo/camel-config

anomalies is sometimes much lower than the number of normal sequences, the performance is measured in metrics suitable for imbalanced data: *precision*, *recall* and *f1-score*. However, a consequence of letting $y = 1$ represent the normal class is that, technically, the metrics used to evaluate the models are *specificity* and *negative predictive value* rather than precision and recall. Since precision, recall, and f1-score are usually used when presenting results for anomaly detection the following definitions of precision, recall and f1-score are used in this thesis to avoid unnecessarily confusing nomenclature:

$$\begin{aligned} \text{precision} &= p(y = 0|\hat{y} = 0), \\ \text{recall} &= p(\hat{y} = 0|y = 0), \\ \text{f1} &= \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \end{aligned}$$

Each experiment was run five times using five different seeds. The seeds used were $1, 12, 123, 1234, 12345$.

Datasets

There are five publicly available labelled datasets available on LogHub [18]. Of the five, the two datasets named HDFS [40] and BGL [28] are most commonly used throughout the literature. Therefore, to make a fair comparison to other models, we use the same datasets. Additionally, a dataset from a real-world site using Sectra's products is used. Statistics about the datasets are displayed in Table 3.1. Each dataset was divided into train, validation and test data. The training, validation and test sets consisted of 60%, 10% and 30% of the dataset respectively. The training set was further divided into two distinct parts according to the PU learning formulation; D_P which is the set of known normal sequences and D_u which is the unlabelled sequences, so that $D_p \cap D_u = \varnothing$ and $D_p \cup D_u = D_t$ where D_t denotes the training set.

Table 3.1: Information about the datasets

Dataset	#Log messages	#Log sequences	#Anomalous sequences
HDFS	11,175,629	575,061	16,838
BGL	4,747,967	85,577	36,303
Sectra System A	3,081,021	154,085	42,603

The HDFS dataset is generated in a private cloud environment using benchmark workloads and manually labelled through handcrafted rules. It was introduced in [40]. The logs are divided into session windows using the block IDs as identifiers. Then each window is assigned a label (normal/anomaly) through the handcrafted rules. Since it is very difficult to write perfect handcrafted rules, the authors note that it is possible that there are mislabelled examples in the dataset. Note that in this dataset the labels are set on session windows rather than individual log messages [40].

BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768 GB of memory. It was introduced by [28] to provide a relevant dataset for log analytics, including log anomaly detection. The log contains alert and non-alert messages identified by alert category tags. The labels were produced by expert system administrators, usually by them creating regular expressions, matching lines that they would categorise as alerts. This labelling procedure means that while all messages marked as anomalies are correct, there might be some messages marked as normal that should be marked as anomalies. In this dataset, each message has an attached label. This means that when classifying a sequence

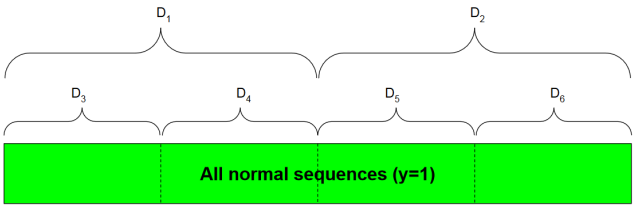


Figure 3.3: Illustration of the six splits of the known normal sequences used in performance evaluation

of log messages, some heuristics must be used in order to determine whether the sequence is anomalous or normal. Normally a strict policy is used, where the sequence is considered normal only if it contains zero anomalous messages. [28]

The final dataset is a real-world dataset provided by Sectra, in this report referred to as *SDBS*. The dataset is a set of log files produced by the database system of a large PACS (picture archiving and communication system) run at a real-world site. Each log message was labelled as normal/anomalous by using hand-crafted regexes currently in use by the Sectra to monitor for anomalies. It is therefore guaranteed that messages labelled as anomalies are in fact anomalies, but the set might contain mislabelled normal sequences. Unfortunately, this dataset cannot be grouped by session window since it lacks the required identifiers. Therefore, the log sequences were grouped using a fixed window of size 20. When grouping the sequences in this dataset chronologically the resulting split is heavily skewed with just a single anomalous instance in the entire validation dataset. For this reason, SDBS was split into train, validation and test data using the first common approach listed in section 2.2; randomly shuffling all sequences and selecting appropriate amounts of them for each dataset.

Prior Optimisation

Since the prior π for *nnPU* and *Imb-nnPU* is treated as a hyperparameter, the first step in the experiments was to determine the optimal value for this parameter for each dataset. The value of π was evaluated in the interval $[0.01, 0.99]$ with a step size of 0.03 for each of the three datasets. Finally, the median results of each prior π were plotted in a graph to determine the optimal value for the hyperparameter.

Performance with varying known sequences

Since all models in this thesis rely on a set of known normal data, D_p, the choice of D_p might impact the performance of the models. Further, a larger size of D_p should produce models performing better. To investigate the impact of D_p simultaneously with investigating the relative performance of the modifications suggested in this thesis, the full set of normal sequences was separated into six parts. The first two splits, D_1 and D_2, are two non-overlapping sets of 50% of the full set of normal sequences. Each of these datasets was then divided in half, producing four more sets of known data D_{3-6}. Figure 3.3 visualises the split. Each model was then trained five times on each of D_{1-6}. The result of this experiment is meant to produce both the best performance of each model given a varying number of known normal sequences, the variance in performance, and the impact of reducing D_p.

Lowering the number of known normals

Since generating the labelled dataset D_p induces a cost in a real-world scenario, it is interesting to investigate how the size of this set impacts the performance of the task. To accomplish this, the size of D_p was progressively lowered from 50% until the performance curve flattened

out at a bottom level. This was done in two steps. In the first step the size of D_P was lowered with relatively large steps in order to get a rough estimate of the performance. Thereafter, the same grid search approach was employed with tighter steps in each area where the performance dropped. Since this is a time-demanding experiment, it was run with only the baseline and the best performant model from the results of the previous experiment described in Section 3.5.

Introducing Anomalies

While it is relatively easy to generate mostly normal sequences in the real world, it is difficult to guarantee that zero anomalies are present in the dataset. It is therefore important to evaluate the model performance if a few anomalies are mistakenly mislabelled as normal. This experiment introduces some labelled anomalies into D_p and studies the impact on the performance of the models. However, this is not straightforward as the datasets contain different types of anomalies and it is probable that introducing some of them has a larger impact than others. To attempt to alleviate the bias produced by this issue four sets of anomalies were extracted and introduced into D_1. The first two, A_{1-2}, contain 5% of the anomalies sampled randomly where $A_1 \cap A_2 = \varnothing$. The second two, A_{3-4}, are each a randomly sampled fifth of A_1 and A_2 respectively. The number of sequences matching 5% and 1% of all anomalies, i.e. the size of A_{1-2} and A_{3-4} can be seen in Table 3.2. Each model was then trained five times on $D_1 \cup A_i, i \in \{1, 2, 3, 4\}$. The result should indicate the robustness of the models, given the presence of a low number of mislabelled anomalies in the set of known normal data.

Table 3.2: The number of anomalous sequences corresponding to 5% and 1% in HDFS and BGL.

Dataset	5%	1%
HDFS	562	112
BGL	1360	272

Efficiency benchmark

One of the primary requirements for a product to be usable in practice is the time it takes to train and use the model to produce predictions. Therefore, the training and prediction time of each method is investigated. The total time of the pipeline is a sum of five steps: preprocessing, vectorisation, PLE, training, and prediction. Since all methods share the same preprocessing step and one of the two vectorisation methods, these steps are measured separately from the full pipeline. The remaining three steps are measured in a continuous execution by loading pre-vectorised sequences from cache. Each measurement is presented as the mean of three runs on BGL and HDFS. The models were trained for ten epochs whereafter labels for the full test set were predicted. Note that the re-labelling time is included in the training time; i.e. PLE only measures the first probabilistic label estimation done.

Performance in relation to other works

The final experiment conducted in this thesis is to compare the results of the models to the performance of other pre-existing models in the field. The performance for the models in related works is taken from [41], as the authors of this paper use the same pre-processing steps as this work. The authors of the paper re-implemented and evaluated the performance of the following methods:

- Unsupervised: Deeplog[12], LogCluster [22] and PCA

- Semi-Supervised: LogAnomaly [25]. Their own method, PLELog, is a PU learning method using a D_p consisting of 50% of the known normal data.
- Fully supervised: LogRobust [42].

In [41], each model was trained several times with different hyperparameters and only the best result for each model was reported. To make a fair comparison, only the best result from the best model for each dataset on the 25% data splits (D_{3-6}) from the performance experiment in Section 3.5 is reported.

4 Results

In this chapter, the performance of the baseline model with and without the presented modifications is displayed. In total four different experiments were run, and are presented in separate subsections below. The first section, section 4.1, displays the performance of *nnPU* and *Imb-nnPU* using different priors. The rest of the report will be using the optimal prior found in this experiment. Section 4.2 does an exhaustive performance control using different amounts of known data and data splits. Following it, section 4.3 displays the result of progressively lowering the size of D_p. Penultimately, section 4.4 shows the robustness of different models when the known normal data is contaminated with some mislabelled anomalies. Finally, section 4.5 rounds out this chapter by presenting the computational complexity of the methods used.

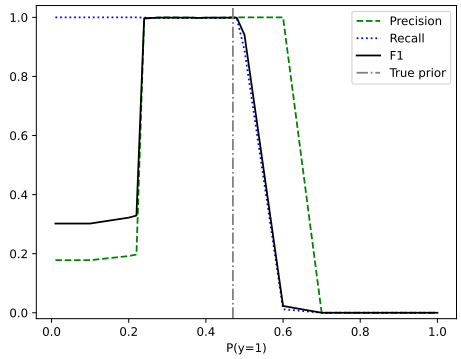
4.1 Prior optimisation

The prior π is important when using empirical risk-minimisation methods. In this section, the performance of *nnPU* and *Imb-nnPU* are evaluated for different values of the prior π, i.e. treating π as a hyperparameter. The result for BGL, HDFS and SDBS are presented as separate subfigures in Figure 4.1. The sequences are vectorised using GloVe embeddings. From the figures, it is clear that both *nnPU* and *Imb-nnPU* are robust to many choices of the prior π. Interestingly, it also shows that using the true value of π does not always produce the best result, indicating that even if π is known it might not be optimal to use it. This is especially clear for *nnPU*, where the true prior is always close to the edge of the flat area of best performance. For *nnPU* the results match the theory; for very high values of π the model predicts all sequences as normal, and for very low values the model predicts all sequences as anomalies. However, for *Imb-nnPU* the performance varies more between datasets. For BGL, the performance is consistently high for all priors up to a certain point ($\pi \approx 0.6$) where the model starts to predict all points as anomalies. As a contrast, the choice of prior seems mostly inconsequential for HDFS, where the variance in performance is almost randomly spread across all priors. Finally, the result for SDBS shows that the performance is continuously good until $\pi \approx 0.6$, where the performance starts to decrease rapidly. For $\pi > 0.9$, the model predicts all sequences as normal, causing a score of zero. Based on these results, the remaining experiments are run using the values for π presented in Table 4.1.

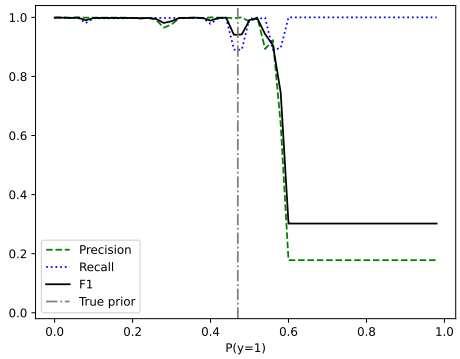
Table 4.1: Optimal priors for *nnPU* and *Imb-nnPU*.

Dataset	*nnPU*	*Imb-nnPU*
BGL	0.40	0.40
HDFS	0.85	0.02
SDBS	0.60	0.20

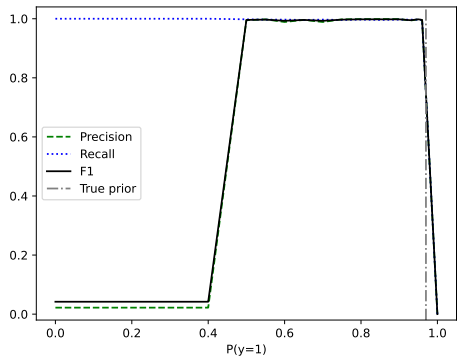
For *nnPU* **For *Imb-nnPU***



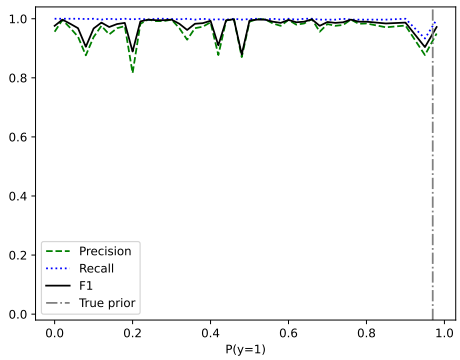
(a) Performance on BGL for varying values of π



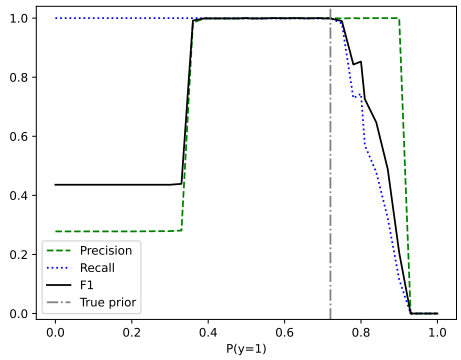
(b) Performance on BGL for varying values of π



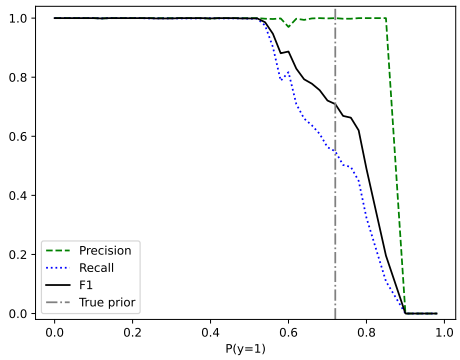
(c) Performance on HDFS for varying values of π



(d) Performance on HDFS for varying values of π



(e) Performance on SDBS for varying values of π



(f) Performance on SDBS for varying values of π

Figure 4.1: Performance with varying priors

4.2 Varying known normals

This section aims to validate the model performance by presenting results on six different splits of the known normal data. The training data was shuffled and split into six parts, where D_1 and D_2 are each half of all normal sequences in the training data and D_{3-6} the four halves of D_1 and D_2. The different models were then trained and evaluated five times on each data split D_i, where each training was done with a different seed. The max, median, min and standard deviation for each data split D_i are displayed in Table 4.2, 4.3 and 4.4. The results from all experiments are available in Appendix B.

One clear trend is visible: convergence on HDFS is less reliable than on the other two datasets, with the results sometimes displaying low F1-scores on the dataset for all D_i while almost no such outlier scores are present in the results from BGL or SDBS. Furthermore, all methods except the baseline perform equally well on BGL for $D_{1,2}$ but the performance drops for all methods except for *nnPU* or *Imb-nnPU* on D_{3-6}.

On HDFS the results are harder to interpret due to the large variance in performance. Two things are clearly visible though: first, using *nnPU* or *Imb-nnPU* outperforms the other models both in variance, mean and max score and secondly, *IR* with GloVe is an improvement over the baseline. Finally, the results indicate that combining *FT* with any other modification results in a worse or equally good performance when compared to the modifications separately, especially for D_{3-6}. Only the baseline is consistently better when combined with *FT*.

On SDBS, in Table 4.4, the scores for the models using *nnPU* or *Imb-nnPU* on $D_3 - D_6$ are much higher and with lower variance than the rest. On D_1 and D_2 the models perform similarly to each other, but *nnPU* and *Imb-nnPU* still outperform the rest.

Table 4.2: Performance for the different models with varying known normals on BGL.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.977	0.998	0.988	0.998	0.998	0.998
0.964	0.981	0.972	0.981	0.977	0.978
0.991	0.910	0.949	1.000	0.890	0.942
0.021	0.035	0.014	0.017	0.043	0.020

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.845	0.998	0.915	0.801	1.000	0.889	0.888	0.999	0.940	0.804	0.998	0.891
0.806	0.949	0.871	0.807	0.937	0.866	0.833	0.937	0.881	0.807	0.917	0.858
0.762	0.913	0.831	0.786	0.897	0.838	0.775	0.896	0.831	0.761	0.898	0.823
0.033	0.042	0.031	0.031	0.049	0.020	0.037	0.050	0.037	0.027	0.041	0.022

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
0.992	0.998	0.995	0.999	0.998	0.999
0.984	0.999	0.991	0.998	0.998	0.998
0.005	0.000	0.002	0.001	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.922	0.998	0.959	0.912	0.998	0.953	0.872	0.998	0.931	0.987	0.997	0.992
0.845	0.999	0.915	0.837	0.999	0.909	0.834	0.998	0.908	0.859	0.998	0.921
0.771	0.999	0.870	0.734	1.000	0.847	0.788	0.998	0.881	0.762	0.999	0.865
0.054	0.000	0.032	0.064	0.001	0.038	0.032	0.000	0.019	0.084	0.001	0.048

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	0.999	0.998	0.999
0.999	0.999	0.999	0.999	0.998	0.998
0.998	0.998	0.998	0.997	0.999	0.998
0.001	0.000	0.000	0.001	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.835	0.999	0.910	0.989	0.999	0.994	0.807	0.998	0.892	0.983	0.998	0.990
0.802	0.999	0.889	0.861	0.998	0.924	0.806	0.897	0.849	0.926	0.892	0.909
0.761	0.999	0.864	0.799	0.998	0.888	0.795	0.957	0.867	0.931	0.977	0.953
0.024	0.000	0.015	0.067	0.000	0.037	0.031	0.050	0.017	0.036	0.042	0.029

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.998	0.999	0.998	0.998
0.997	0.998	0.998	0.995	0.998	0.996
0.001	0.000	0.000	0.002	0.000	0.001

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.998	0.998	0.939	0.999	0.968	0.991	0.998	0.994	1.000	0.998	0.999
0.953	0.998	0.974	0.903	0.999	0.947	0.930	0.976	0.947	0.984	0.998	0.991
0.833	0.998	0.908	0.803	0.999	0.890	0.710	0.999	0.830	0.955	0.999	0.977
0.061	0.001	0.034	0.051	0.001	0.029	0.111	0.043	0.062	0.015	0.001	0.008

Table 4.2: Performance for the different models with varying known normals on BGL, continued.

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.998	1.000	0.998	0.999
0.994	0.998	0.996	0.999	0.998	0.999
0.983	0.998	0.991	0.999	0.998	0.999
0.006	0.000	0.003	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.998	0.998	0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999
0.985	0.977	0.981	0.996	0.998	0.997	0.999	0.998	0.999	1.000	0.998	0.999
0.983	0.890	0.934	0.983	0.998	0.991	0.997	0.998	0.998	1.000	0.997	0.998
0.006	0.043	0.023	0.006	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.998	0.998	0.998	1.000	0.998	0.999
0.001	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.993	0.998	0.996	0.999	0.998	0.998	0.999	0.998	0.999	0.999	0.998	0.999
0.984	0.998	0.991	0.996	0.998	0.997	0.998	0.998	0.998	0.999	0.997	0.998
0.007	0.000	0.004	0.001	0.000	0.001	0.001	0.000	0.000	0.000	0.000	0.000

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.998	0.998	0.998	1.000	0.998	0.999
0.001	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.999	0.999	1.000	0.998	0.999
0.990	0.999	0.994	0.999	0.998	0.999	0.999	0.999	0.999	1.000	0.998	0.999
0.982	0.999	0.991	0.998	0.998	0.998	0.999	0.998	0.999	1.000	0.998	0.999
0.007	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.000	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.996	0.998	0.997	0.999	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.998
0.984	0.998	0.991	0.999	0.998	0.999	1.000	0.998	0.999	0.996	0.997	0.997
0.006	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.001

Table 4.3: Performance for the different models with varying known normals on HDFS.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.942	0.970	0.996	0.918	0.955
0.784	0.933	0.819	0.848	0.902	0.847
0.330	0.956	0.491	0.326	0.891	0.477
0.276	0.020	0.188	0.262	0.010	0.185

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.996	0.945	0.970	0.999	0.963	0.981	0.998	0.956	0.977	0.999	0.896	0.945
0.978	0.950	0.963	0.894	0.955	0.917	0.935	0.946	0.938	0.853	0.908	0.853
0.960	0.948	0.954	0.631	0.965	0.763	0.757	0.948	0.842	0.344	0.918	0.501
0.018	0.012	0.006	0.136	0.020	0.078	0.093	0.009	0.051	0.255	0.019	0.176

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.992	0.976	0.984	0.991	0.967	0.979
0.984	0.957	0.970	0.987	0.930	0.958
0.973	0.945	0.959	0.975	0.909	0.941
0.010	0.014	0.008	0.007	0.020	0.013

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.992	0.979	0.986	1.000	0.976	0.988	0.991	0.978	0.984	0.987	0.966	0.976
0.983	0.955	0.968	0.995	0.972	0.983	0.994	0.953	0.973	0.828	0.936	0.844
0.970	0.949	0.959	0.981	0.975	0.978	0.992	0.936	0.963	0.258	0.917	0.403
0.017	0.015	0.010	0.007	0.006	0.004	0.004	0.015	0.008	0.286	0.020	0.221

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.998	0.995	0.997	0.986	0.997	0.991
0.964	0.979	0.971	0.788	0.986	0.822
0.944	0.963	0.953	0.123	0.963	0.219
0.025	0.016	0.019	0.333	0.013	0.302

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.997	0.970	0.983	0.997	0.977	0.987	1.000	0.967	0.983	0.995	0.900	0.945
0.804	0.931	0.800	0.983	0.953	0.968	0.812	0.953	0.816	0.925	0.912	0.915
0.114	0.961	0.204	0.930	0.877	0.903	0.108	0.941	0.194	0.804	0.976	0.882
0.346	0.038	0.299	0.026	0.039	0.032	0.352	0.011	0.311	0.072	0.038	0.024

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.869	0.992	0.926	0.951	0.997	0.973
0.587	0.957	0.674	0.785	0.950	0.827
0.095	0.978	0.174	0.213	0.878	0.343
0.310	0.052	0.296	0.286	0.040	0.242

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.733	0.859	0.791	0.095	0.971	0.173	0.954	0.927	0.940	0.985	0.767	0.862
0.471	0.855	0.531	0.087	0.892	0.158	0.359	0.892	0.421	0.255	0.812	0.279
0.077	0.775	0.141	0.080	0.820	0.146	0.069	0.880	0.128	0.063	0.793	0.116
0.318	0.067	0.309	0.005	0.052	0.010	0.360	0.036	0.343	0.365	0.045	0.292

Table 4.3: Performance for the different models with varying known normals on HDFS, continued.

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.997	0.998	0.996	0.997	0.997
0.999	0.996	0.997	0.944	0.997	0.969
0.999	0.995	0.997	0.849	0.998	0.917
0.000	0.001	0.000	0.055	0.000	0.030

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.997	0.998	0.999	0.996	0.997	0.998	0.996	0.997	0.996	0.997	0.996
0.997	0.994	0.995	0.996	0.996	0.996	0.998	0.994	0.996	0.993	0.997	0.995
0.998	0.986	0.992	0.988	0.997	0.993	0.998	0.992	0.995	0.988	0.998	0.993
0.004	0.004	0.002	0.004	0.001	0.002	0.001	0.002	0.001	0.003	0.001	0.001

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.931	0.964	0.995	0.930	0.962
0.999	0.930	0.963	0.998	0.906	0.949
0.999	0.930	0.963	0.999	0.878	0.935
0.000	0.001	0.000	0.002	0.023	0.012

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.931	0.964	0.998	0.930	0.963	0.999	0.930	0.963	0.997	0.913	0.953
0.995	0.920	0.956	0.995	0.926	0.959	0.999	0.919	0.957	0.997	0.892	0.941
0.998	0.881	0.936	0.980	0.915	0.947	0.999	0.880	0.936	0.994	0.881	0.934
0.008	0.020	0.011	0.007	0.006	0.006	0.000	0.019	0.011	0.002	0.014	0.008

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.993	0.998	0.995	0.997	0.997	0.997
0.919	0.998	0.954	0.790	0.997	0.814
0.732	0.998	0.845	0.076	0.998	0.141
0.098	0.000	0.057	0.359	0.000	0.337

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.997	0.998	0.998	0.997	0.997	0.995	0.997	0.996	0.997	0.997	0.997
0.980	0.997	0.988	0.993	0.997	0.995	0.677	0.996	0.752	0.979	0.997	0.987
0.962	0.996	0.979	0.985	0.998	0.991	0.253	0.997	0.404	0.920	0.997	0.957
0.012	0.001	0.006	0.005	0.001	0.002	0.334	0.001	0.269	0.030	0.000	0.016

(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.996	0.929	0.962	0.716	0.945	0.815
0.916	0.932	0.923	0.570	0.935	0.690
0.870	0.935	0.901	0.257	0.933	0.403
0.043	0.002	0.020	0.171	0.018	0.153

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.994	0.930	0.961	0.987	0.930	0.957	0.965	0.932	0.948	0.999	0.881	0.936
0.519	0.937	0.606	0.801	0.935	0.816	0.840	0.909	0.861	0.759	0.896	0.781
0.224	0.952	0.362	0.186	0.950	0.310	0.569	0.904	0.698	0.253	0.889	0.394
0.337	0.008	0.266	0.309	0.008	0.253	0.180	0.026	0.100	0.301	0.026	0.212

Table 4.4: Performance for the different models with varying known normals on SBDS.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.981	0.935	0.957	0.942	0.953	0.947
0.904	0.925	0.912	0.889	0.919	0.903
0.737	0.927	0.821	0.836	0.915	0.874
0.089	0.014	0.047	0.046	0.028	0.026

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.440	0.966	0.605	0.442	0.966	0.607	0.424	0.901	0.576	0.437	0.899	0.588
0.416	0.927	0.575	0.411	0.930	0.570	0.404	0.929	0.563	0.409	0.926	0.567
0.407	0.909	0.563	0.398	0.917	0.555	0.391	0.941	0.552	0.393	0.940	0.554
0.013	0.023	0.015	0.017	0.034	0.019	0.011	0.026	0.009	0.016	0.017	0.013

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.957	0.956	0.957	0.987	0.961	0.974
0.893	0.955	0.922	0.929	0.950	0.938
0.802	0.944	0.867	0.816	0.946	0.876
0.052	0.008	0.030	0.066	0.011	0.035

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.428	0.955	0.591	0.438	0.953	0.600	0.424	0.969	0.589	0.422	0.944	0.583
0.415	0.967	0.580	0.415	0.962	0.579	0.412	0.963	0.577	0.410	0.954	0.573
0.409	0.970	0.575	0.400	0.983	0.569	0.401	0.955	0.565	0.400	0.957	0.564
0.007	0.007	0.006	0.013	0.011	0.011	0.010	0.009	0.010	0.008	0.005	0.007

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	0.999	1.000	0.999
0.933	0.996	0.962	0.997	0.996	0.997
0.831	0.989	0.903	0.999	0.986	0.992
0.067	0.004	0.038	0.004	0.005	0.003

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.879	0.996	0.934	0.873	0.977	0.922	0.876	0.992	0.930	0.970	0.987	0.978
0.754	0.949	0.838	0.757	0.930	0.830	0.808	0.939	0.863	0.781	0.957	0.857
0.615	0.916	0.736	0.712	0.776	0.743	0.647	0.997	0.785	0.680	0.979	0.802
0.095	0.042	0.070	0.089	0.080	0.068	0.087	0.065	0.047	0.099	0.031	0.063

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.996	0.995	0.996	0.999	0.999	0.999
0.972	0.990	0.981	0.993	0.996	0.994
0.957	0.989	0.973	0.980	0.998	0.989
0.019	0.010	0.008	0.008	0.003	0.004

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.723	1.000	0.839	0.776	0.958	0.857	0.844	0.926	0.883	0.812	0.987	0.891
0.714	0.923	0.797	0.807	0.825	0.801	0.690	0.935	0.789	0.786	0.900	0.818
0.583	0.984	0.733	0.799	0.718	0.756	0.593	1.000	0.744	0.954	0.566	0.710
0.079	0.094	0.037	0.102	0.130	0.037	0.089	0.053	0.050	0.105	0.167	0.068

Table 4.4: Performance for the different models with varying known normals on SBDS, continued.

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	1.000	1.000	1.000	0.999	1.000
0.999	1.000	0.999	1.000	0.999	1.000
0.999	1.000	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	1.000	0.999	0.999	1.000	0.999	0.999	1.000	0.999	1.000
0.999	1.000	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.998	1.000	0.999	1.000	0.998	0.999	0.999	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	0.999	0.999	0.999	0.999
0.999	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	1.000	0.999	0.999	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	1.000	0.999	1.000
1.000	1.000	1.000	0.999	0.999	0.999
1.000	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	0.999	1.000	0.999	1.000	0.999	1.000	1.000	0.999	1.000
1.000	0.999	1.000	0.999	1.000	0.999	1.000	0.999	1.000	1.000	0.999	0.999
1.000	0.999	0.999	0.998	1.000	0.999	1.000	0.999	0.999	0.998	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.000

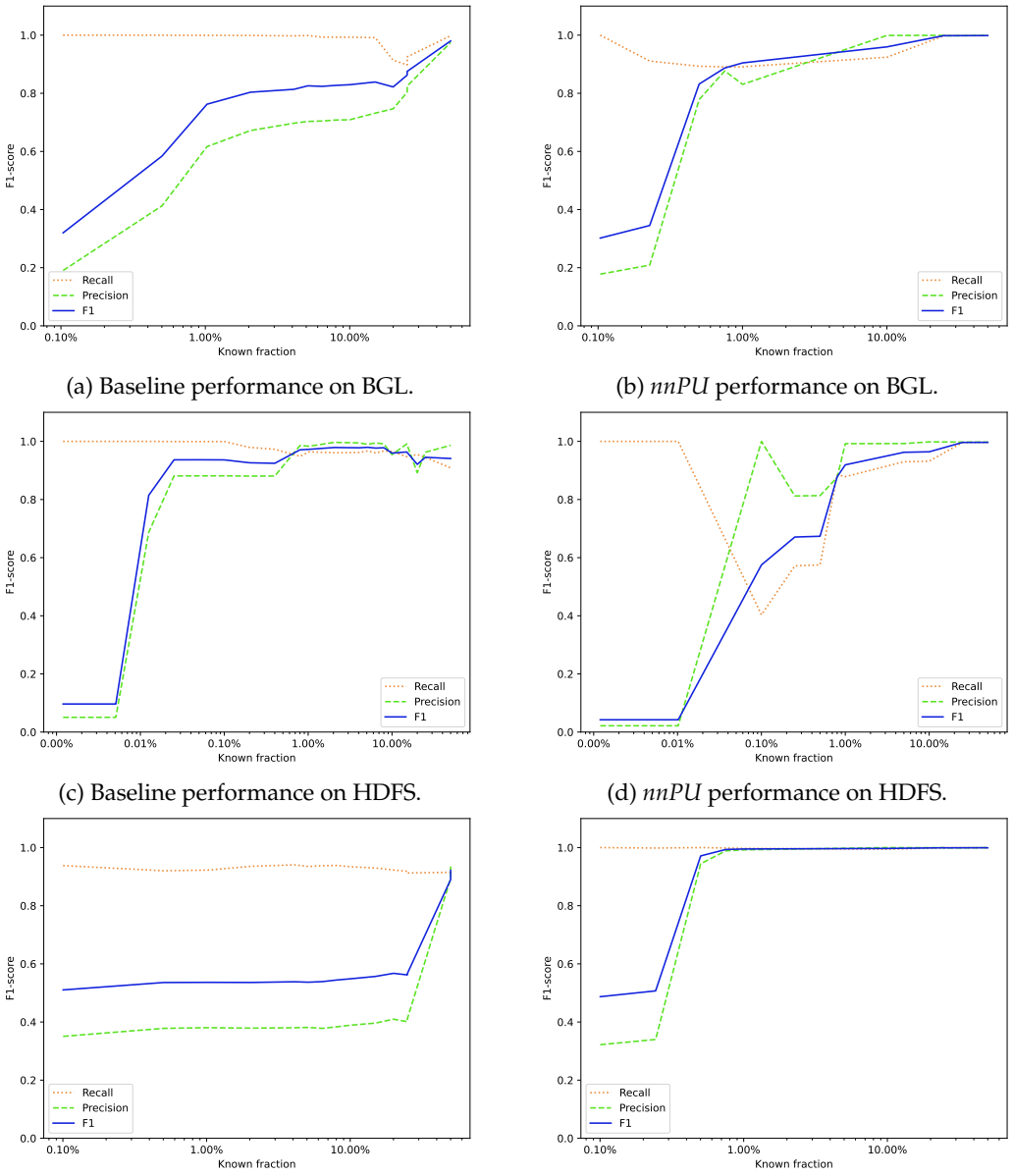
(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	0.999	0.999
0.999	1.000	0.999	0.999	1.000	0.999
0.000	0.000	0.000	0.000	0.000	0.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	1.000	0.999	0.999
1.000	0.999	1.000	0.998	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	1.000	1.000	0.998	1.000	0.999	1.000	0.998	0.999	1.000	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

4.3 Lowering the number of known normals

The experiments in this section study the impact of incrementally decreasing the size of D_P from 50% until the performance flattened out at a bottom level. The minimum size of D_p was 0.001% for HDFS and 0.01% for BGL and SDBS. The experiments were conducted with the baseline and the *nnPU* model. As shown in Figure 4.2, the baseline performs well on BGL and SDBS when 50% of the normal sequences are known but drops quickly after that. In contrast, *nnPU* and *Imb-nnPU* perform well until it the size of D_p goes below 0.4% for BGL and 0.6% for SDBS. However, this relationship is flipped for HDFS where the baseline consistently has higher than 0.9 in F1-score when D_P is larger than 0.01%, while the F1-score for *nnPU* reduces steadily while reducing D_P. Finally, both models start to predict most samples as anomalies when the size of D_P is 0.01% of the set of known sequences.



(a) Baseline performance on BGL.

(b) *nnPU* performance on BGL.

(c) Baseline performance on HDFS.

(d) *nnPU* performance on HDFS.

(e) Baseline performance on SDBS.

(f) *nnPU* on SDBS.

Figure 4.2: Lowering the number of known normals. The lines display the median value. Note the different scale on the x-axis for (c) and (d).

4.4 Introducing Anomalies

In this experiment, anomalies were intentionally mislabelled and injected into D_P. The resulting performance for BGL and HDFS are presented in Table 4.5 and Table 4.6 respectively. The rows displayed are max, median, min and standard deviation. The results indicate that the methods which rely on pseudo-labels suffer performance decreases with just 1% of the anomalies introduced. This is especially true for the models incorporating iterative relabelling, where the performance decrease is notably larger when compared to data without introduced anomalies. However, the performance of *nnPU* and *Imb-nnPU* degrades only a negligible amount for any amount of introduced anomalies. The models implementing the probabilistic labelling technique are shown to have a low tolerance for mislabelled anomalies. However, simple methods can be implemented to alleviate this, and one such method can be seen in appendix C.

Table 4.5: Introducing anomalies BGL with no threshold

(a) Baseline.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.905	0.727	0.806	0.874	0.999	0.932
0.790	0.726	0.755	0.852	0.999	0.920
0.761	0.654	0.703	0.795	0.729	0.761
0.062	0.029	0.036	0.027	0.134	0.077
	A_3			A_4	
0.885	0.999	0.938	0.831	0.998	0.907
0.864	0.998	0.918	0.822	0.998	0.902
0.746	0.611	0.672	0.759	0.238	0.362
0.061	0.165	0.101	0.028	0.296	0.210

(b) Baseline with FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.907	0.996	0.949	0.939	0.999	0.968
0.855	0.721	0.738	0.926	0.723	0.812
0.710	0.721	0.715	0.833	0.651	0.731
0.069	0.164	0.101	0.038	0.149	0.093
	A_3			A_4	
0.958	0.652	0.776	0.904	0.998	0.949
0.814	0.601	0.692	0.812	0.580	0.681
0.795	0.266	0.398	0.812	0.236	0.366
0.073	0.157	0.133	0.060	0.276	0.214

(c) Iterative Relabelling with GloVe embeddings.

(d) Iterative Relabelling with FastText embeddings

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.938	0.922	0.930	0.982	0.722	0.832
0.859	0.730	0.805	0.968	0.722	0.826
0.859	0.392	0.539	0.719	0.504	0.592
0.038	0.207	0.140	0.099	0.090	0.094
	A_3			A_4	
0.990	0.942	0.965	0.884	0.999	0.938
0.834	0.942	0.867	0.884	0.727	0.831
0.783	0.416	0.543	0.677	0.403	0.505
0.078	0.221	0.151	0.098	0.234	0.173

(e) nnPU with GloVe embeddings

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.928	0.725	0.814	0.987	0.998	0.993
0.731	0.599	0.658	0.893	0.723	0.831
0.620	0.277	0.383	0.783	0.545	0.643
0.100	0.154	0.141	0.082	0.177	0.126
	A_3			A_4	
0.940	0.697	0.800	0.883	0.655	0.752
0.829	0.660	0.772	0.783	0.566	0.644
0.732	0.484	0.583	0.702	0.212	0.325
0.101	0.085	0.084	0.061	0.173	0.157

(f) nnPU with FastText embeddings

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
1.000	0.998	0.999	0.999	0.998	0.999
0.998	0.998	0.998	0.999	0.998	0.999
0.998	0.998	0.998	0.996	0.998	0.997
0.001	0.000	0.000	0.001	0.000	0.001
	A_3			A_4	
0.999	0.998	0.999	1.000	0.998	0.999
0.997	0.998	0.998	0.999	0.998	0.999
0.996	0.998	0.997	0.996	0.998	0.997
0.001	0.000	0.001	0.001	0.000	0.001

(g) Imbalanced nnPU with GloVe embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.998	0.998	0.998	1.000	0.998	0.999
0.001	0.000	0.000	0.000	0.000	0.000
	A_3			A_4	
0.999	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	0.998	0.998	0.998
0.000	0.000	0.000	0.001	0.000	0.000

(h) Imbalanced nnPU with FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
1.000	0.998	0.999	0.999	0.998	0.999
0.998	0.998	0.998	0.998	0.998	0.998
0.998	0.998	0.998	0.996	0.998	0.997
0.001	0.000	0.000	0.001	0.000	0.001
	A_3			A_4	
1.000	0.998	0.999	1.000	0.998	0.999
0.998	0.998	0.998	0.998	0.998	0.998
0.996	0.998	0.997	0.997	0.998	0.998
0.001	0.000	0.001	0.001	0.000	0.000

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.000	0.000	0.000	0.000	0.000	0.000
	A_3			A_4	
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	0.999	0.998	0.999
0.000	0.000	0.000	0.000	0.000	0.000

Table 4.6: Introducing anomalies HDFS

(a) Baseline.

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.967	0.040	0.076	0.877	0.065	0.121
0.967	0.028	0.055	0.941	0.060	0.114
0.987	0.020	0.039	0.941	0.030	0.058
0.064	0.008	0.015	0.048	0.015	0.028
A_3			A_4		
0.893	0.138	0.240	0.990	0.106	0.192
0.906	0.116	0.206	0.983	0.085	0.157
0.587	0.099	0.170	0.966	0.045	0.086
0.143	0.016	0.027	0.023	0.021	0.037

(b) FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.555	0.053	0.097	0.396	0.083	0.137
0.895	0.021	0.042	0.994	0.020	0.040
0.964	0.007	0.014	1.000	0.011	0.021
0.163	0.018	0.032	0.234	0.027	0.043
A_3			A_4		
0.908	0.106	0.189	0.850	0.094	0.169
0.908	0.097	0.151	0.850	0.057	0.107
0.042	0.111	0.061	0.627	0.040	0.075
0.361	0.020	0.048	0.124	0.024	0.041

(c) Iterative with GloVe embeddings

(d) Iterative with FastText embeddings

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.869	0.355	0.504	0.744	0.348	0.474
0.869	0.314	0.447	0.744	0.303	0.435
0.948	0.258	0.406	0.773	0.249	0.377
0.054	0.037	0.038	0.057	0.032	0.034
A_3			A_4		
0.909	0.531	0.671	0.873	0.567	0.688
0.909	0.512	0.601	0.873	0.536	0.667
0.416	0.495	0.452	0.853	0.536	0.658
0.242	0.033	0.090	0.040	0.019	0.011

(e) nnPU with GloVe embeddings

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.750	0.330	0.458	0.836	0.265	0.403
0.624	0.327	0.411	0.267	0.261	0.250
0.064	0.504	0.113	0.056	0.261	0.092
0.305	0.096	0.144	0.300	0.094	0.108
A_3			A_4		
0.907	0.508	0.651	0.941	0.522	0.671
0.885	0.430	0.567	0.739	0.492	0.570
0.885	0.387	0.538	0.637	0.442	0.522
0.130	0.053	0.040	0.189	0.053	0.063

(f) nnPU with FastText embeddings

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.998	0.996	0.997	0.999	0.997	0.998
0.998	0.996	0.997	0.999	0.996	0.997
0.982	0.997	0.989	0.993	0.995	0.994
0.006	0.001	0.003	0.002	0.001	0.001
A_3			A_4		
0.999	0.997	0.998	0.999	0.997	0.998
0.998	0.997	0.997	0.999	0.996	0.997
0.947	0.998	0.972	0.998	0.996	0.997
0.021	0.001	0.010	0.000	0.001	0.000

(g) Imbalanced nnPU with GloVe embeddings.

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.979	0.924	0.951	0.986	0.929	0.957
0.979	0.909	0.944	0.986	0.904	0.949
0.922	0.909	0.915	0.933	0.881	0.906
0.028	0.007	0.014	0.024	0.015	0.018
A_3			A_4		
0.995	0.929	0.961	0.999	0.911	0.953
0.979	0.909	0.939	0.998	0.907	0.935
0.960	0.882	0.919	0.914	0.911	0.913
0.014	0.017	0.016	0.034	0.015	0.014

(h) Imbalanced nnPU with FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.998	0.997	0.997	0.998	0.997	0.997
0.978	0.997	0.987	0.964	0.997	0.981
0.911	0.998	0.952	0.782	0.998	0.877
0.031	0.000	0.017	0.083	0.000	0.046
A_3			A_4		
0.995	0.996	0.995	0.998	0.997	0.998
0.986	0.997	0.992	0.990	0.997	0.993
0.946	0.997	0.971	0.691	0.998	0.817
0.017	0.001	0.009	0.121	0.001	0.071

Precision	Recall	F1	Precision	Recall	F1
A_1			A_2		
0.996	0.932	0.963	0.983	0.932	0.957
0.982	0.932	0.956	0.926	0.932	0.930
0.089	0.962	0.163	0.035	0.978	0.067
0.367	0.012	0.312	0.366	0.018	0.350
A_3			A_4		
0.961	0.950	0.955	0.996	0.931	0.962
0.935	0.942	0.938	0.957	0.932	0.944
0.336	0.945	0.496	0.131	0.945	0.230
0.248	0.014	0.179	0.327	0.007	0.280

4.5 Efficiency benchmark

In Table 4.8 the efficiency of each model with respect to computer time is displayed. The first half was run on BGL and the second half on HDFS. The vectorisation times are for the full dataset, the PLE time is for the training set, the training was done for ten epochs, and the prediction time is for the full test split. The results indicate that the time cost to train a FastText model is significant. However, using the FastText embeddings reduces the convergence time for the neural models by roughly 30% when compared to GloVe embeddings. Similarly, the prediction times are reduced by roughly 30% when using FastText embeddings.

Table 4.7: Time for pre-processing and vectorisation for each dataset in seconds. Includes training and/or loading models.

Dataset	Drain (s)	FastText (s)	GloVe (s)
BGL	321	1041	129
HDFS	390	1133	132

Table 4.8: Time taken for each model. Results over the dashed lines are on BGL and below on HDFS.

Method	PLE (s)	Training (s/epoch)	Prediction time (s)
Baseline	70.6	40.8	7.0
IR	70.2	58.8	7.1
nnPU	0	45.2	7.6
Imb-nnPU	0	46.2	7.5
FT	70.5	30.0	6.2
FT + IR	61.7	45.4	6.1
FT + nnPU	0	30.0	6.2
FT + Imb-nnPU	0	30.9	6.2
Baseline	243.7	172.7	33.9
IR	250.6	260.6	32.6
nnPU	0	156.0	32.9
Imb-nnPU	0	159.0	32.8
FT	240.1	120.6	29.7
FT + IR	227.0	235.6	29.8
FT + nnPU	0	117.8	28.5
FT + Imb-nnPU	0	121.8	28.6

4.6 In relation to other works

In Table 4.9 the performance of the best model investigated in this thesis is compared to the performance of other related models in the field. Extensions provided in this thesis are shown in the table by italicised text. From Table 4.9 it is visible that all extensions produce equally high F1-scores on BGL, and that this score is higher than all other unsupervised models and equally good as the supervised method LogRobust [42]. Further, *nnPU* outperforms all models on HDFS, providing a new state-of-the-art performance on the dataset.

Table 4.9: Performance of the best method in this work together with other works in the field as reported in [41].

(a) On HDFS.

Method	Precision	Recall	F1
DeepLog	0.945	0.900	0.922
LogAnomaly	0.860	0.898	0.878
LogCluster	**1.000**	0.836	0.911
PCA	0.347	0.073	0.121
PLELog	0.950	0.963	0.957
nnPU	0.999	**0.997**	**0.998**
LogRobust	0.999	0.995	0.997

(b) On BGL.

Method	Precision	Recall	F1
DeepLog	0.138	0.63	0.227
LogAnomaly	0.179	0.998	0.303
LogCluster	0.914	0.642	0.754
PCA	0.448	0.333	0.382
PLELog	0.965	**0.999**	0.982
FT+(*IR*/*nnPU*/*Imb-nnPU*), *nnPU* and *Imb-nnPU*	**1.000**	0.998	**0.999**
LogRobust	0.999	0.998	0.999

5 Discussion

This chapter provides a discussion of the work done in this thesis. First, the results and implications of them are discussed in section 5.1. Following it in section 5.2, the limitations of the method and possible improvements to it are discussed. Finally, section 5.3 ends the chapter by discussing the work in a wider context, focusing on the societal, ethical, and environmental impact of the work done in this thesis.

5.1 Results

The results presented in chapter 4 indicate that the annotated anomalies in the datasets investigated can be reliably and accurately detected using the deep learning systems employed in this thesis. Further, it indicates that the labelled benchmark datasets are a good proxy for real-world datasets since models showing a good performance on the benchmark datasets also show a good performance on the real-world dataset. However, some concerns are also evident such that the models perform unreliably on the HDFS dataset, and sometimes resort to classifying all sequences as anomalous when the number of known sequences is too low. Finally, the results from the models studied in this thesis suggest that another newer benchmark dataset should be used in place of BGL since the F1-score on the dataset for many of the models investigated was 0.999, and these models are therefore indistinguishable from each other on the dataset.

Applying PU learning to log anomaly detection

The results conclusively indicate that the attempted empirical risk minimisation methods are applicable to log anomaly detection, with *nnPU* and *Imb-nnPU* producing the best results for the methods studied for both the benchmark datasets and the real-world dataset. However, a weakness reveals itself in section 4.2 where the models sometimes for no easily discernable reason produce much lower F1-scores on the HDFS dataset. This trend is especially visible when using *Imb-nnPU*, but applies to a lesser extent to *nnPU* as well. This unstable behaviour is troubling if the model were to be applied in the real world, especially if it is difficult to validate the model performance due to a lack of labelled data. It is lessened by the fact that it is easy to see if the model training has been successful since the models performing worse

induce a very high number of false alarms. Since the unstable behaviour occurs only rarely, retraining the model is likely to correct the error. Still, an interesting extension of this thesis would be investigating the cause of the strange behaviour to find a way to avoid the problem altogether.

Finally, the performance on BGL and SDBS for varying values of the prior π displays an interesting phenomenon for *Imb-nnPU*; when the prior goes above ≈ 0.6 on BGL the model predicts all samples as anomalies, but for SDBS it instead predicts all samples as normal. The behaviour of the model on SDBS is expected as a large π should result in a bias towards predicting normal sequences. The strange contrasting behaviour for BGL might be explained by the fact that after the labelling process, the majority class is in fact $y = 0$, the anomaly class. Therefore, a possible explanation is that the re-weighting scheme introduced in *Imb-nnPU* is due to the model predicting the majority class when the prior π reaches a sufficiently large value.

Model performance

The most noteworthy result from section 4.2 is that *nnPU* and *Imb-nnPU* are almost completely resistant to the reduced number of known normal sequences studied in that section on BGL and SDBS, consistently scoring F1-score higher than 0.997 on all D_{1-6} except for D_3. An explanation of the resiliency in the face of reduced data is that when using the alternate risk function the pipeline no longer relies on pseudo-labelling. Since the pseudo-labels are based on identifying normal and anomalous clusters, a smaller D_p will likely cause full clusters of normal sequences to be misidentified as anomalous. This can have a heavy impact, where large clusters containing numerous normal sequences get mislabelled as anomalies. This would cause an increase in false alarms, which should produce a lower precision but a consistently high recall. This trend is clearly visible through all tables using pseudo-labels; the reduction in performance when going from 50% to 25% of known data is primarily driven by a reduced precision rather than a decreased recall. This is obviously unwanted for an anomaly detection model since frequent false alarms would increase the workload of the engineers receiving the alerts.

The problem of frequent false alarms on BGL and SBDS for smaller D_p was shown to be alleviated by using iterative relabelling (*IR*), while no such distinction can be made on HDFS. One possible reason for this effect is that the re-labelling allows for the correction of obvious mislabelled sequences before the model has been forced to overfit to the incorrectly labelled dataset. However, the performance and robustness increase achieved on BGL is at the cost of a much longer training time as seen in section 4.5.

Finally, while using *FT* lowered the performance of the methods on HDFS, it significantly reduced the training and prediction times as seen in Table 4.8. This is likely due to the smaller embedding dimension of the FastText embeddings. Therefore, for tasks where efficiency is crucial, such as for high-velocity data streams FastText is preferable to the GloVe embeddings used as the performance can likely be kept while significantly reducing the training and prediction times.

Lowering the number of known normals

When the size of D_p is lowered the same pattern is visible for *nnPU* on all three datasets: it has a higher F1-score than the baseline until the performance suddenly drops drastically. Additionally, the results indicate that even with a relatively small amount of known normals the performance will stay high. This is advantageous since labelling data can be time-consuming and expensive. As for the baseline, the performance drops at a more steady pace for BGL and SDBS, but not for HDFS, for which it performs well with as little as 0.1% of D_P known. One

possible reason for the baseline performing so well on HDFS and not for BGL and the SDBS might be that if the sequences in HDFS are similar, then few samples are required to recognise and correctly label the majority of the data. If this is the case, the probabilistic labelling step will correctly label a majority of the sequences even when few sequences are known. This is in contrast to *nnPU* where the loss term related to D_u will force an increased loss when correctly predicting these unknown normal sequences. One possible solution for this problem would be to employ self-supervised learning as in [8, 11], allowing the model to move easily distinguishable examples from D_u to a self-labelled distribution.

Robustness

Section 4.4 displays the results of the models when a certain amount of mislabelled anomalous sequences were introduced into D_p. This result is important when considering the application of log anomaly detection in the industry as it is not unlikely that log files generated by simulating normal behaviour still may contain a low amount of anomalous behaviour, and manually removing such anomalies might be expensive. The results show that mixing a few anomalies in D_p is highly detrimental for the methods relying on pseudo-labels, with the performance suffering greatly with just a small amount of introduced anomalies. However, by introducing a simple threshold algorithm to the pseudo-labelling, the problem was heavily reduced. This indicates that while naively applying the original method of pseudo-labelling is unsuited for practical application, with minor modifications to the pseudo-labelling it might still be useful in a real-world use case where some anomalies are mistakenly labelled as normal.

Additionally, *nnPU* and *Imb-nnPU* proved to be more robust to anomalies being introduced into D_p than the pseudo-labelling methods. This is likely due to the fact that introduced anomalies in D_p have the risk of affecting the labels of many more sequences when estimating probabilistic labels, while mislabelled anomalies using *nnPU* and *Imb-nnPU* are always contained to a single mislabelled sequence. Interestingly, while the overall performance was decreased for *nnPU* and *Imb-nnPU* when anomalies were introduced, none of the evaluations show the outlier effect for HDFS present in Table 4.3. Although this cannot be determined from the experiments performed, this might be due to a stabilising effect of introducing a low number of mislabelled sequences to the training set.

Finally, the same pattern as discussed previously in section 4.2 is present in this experiment, where the performance on HDFS is both worse in general and the difference between best and worst results are larger than on the other datasets for all models. This is especially true for the non-iterative pseudo-labelling strategies whose performances on HFDS are catastrophic. A likely cause of this behaviour is the fact that the iterative re-labelling strategy allows the model to relabel obviously incorrectly labelled examples caused by the introduced anomalies, and therefore mitigate the negative effect of them.

5.2 Method

In this section, a discussion of the method used in this work is presented. The first subsection discusses the method used to evaluate the models, followed by the second subsection which discusses the choice of baseline. The third subsection discusses the interpretability problem for the models investigated in this work. This section is finished with a discussion of the sources used in this thesis.

Model Evaluation

In log anomaly detection, accurate ground-truth labels are essential. In this thesis we have assumed during the evaluation that the ground-truth labels are both accurate and exhaustive; i.e. all reported anomalies are true anomalies, and no anomalous sequences have been mislabelled. While this is a reasonable assumption to make, the creators of the datasets are careful to note that it might not be true. This means that large gaps in performance can be confidently used to decide whether a model is better or worse than another, but when two models perform similarly it is difficult to determine if the lower performance is due to correctly predicting mislabelled examples or incorrectly predicting correctly labelled examples. In fact, the better performing model might have to overfit to incorrect labels, which is unwanted. Thus, the 50%-splits of BGL and SDBS offer few conclusions since all models perform almost equally well.

Further, the process of using regular-expression-based labels when evaluating the models also means that the models are indirectly measured on their ability to replicate the results from the regular expressions, not detecting anomalies. Thus, models performing near 1 in F1-score have failed to detect eventual mislabelled anomalies, indicating that they will also fail to detect new anomalies. If this is the case, it questions whether automated log anomaly detection models are better than using handwritten regular expressions for anomaly detection to begin with. A better evaluation metric would be to through the help of domain experts evaluate each predicted anomaly and let them give feedback on whether the detected anomaly should be reported or not. However, this requires both root-cause analysis to give the domain experts a reasonable workload, and significant time commitment from human personnel giving the feedback. Finally, an alternate method of evaluating the adaptability of the models when faced with previously unseen anomalies using a dataset labelled with regular expressions is to remove all sequences matching a given regular expression from the training data, thereby ensuring that some anomalous sequences in the test data are previously unseen. These hidden sequences can then be used to study whether the model has the capability to detect previously unseen anomalies. If evaluating in this way special care must be taken to ensure that the 'hidden' sequences in the test data do not match another anomaly pattern that the model has seen since that would let the model correctly classify it for the wrong reasons.

Baseline choice

The task of log anomaly detection requires a long pipeline from parsing the raw log file to doing the binary prediction task. During each step on the way, there is no shortage of different methods to use. Regretfully, this means that most of the steps must be taken on faith based on previous works. We have chosen to base our model on a slightly modified version of [41]. The reason for this is that it is to our best knowledge the state-of-the-art model using deep learning methods and given the PU learning scenario. However, this necessarily means that results in this work indicating that a certain modification increases the performance on the downstream task are only valid when compared to the baseline pipeline. No deduction can be made from this work about whether a certain modification is better *in general*.

Interpretability

A major limitation of this study is the choice of not studying the interpretability of the results. All models studied in this thesis stop at reporting a log sequence as anomalous and leave the question of *why* this particular sequence is anomalous open. This is partly a consequence of root-cause analysis being outside the scope of this thesis, but also a problem with using deep learning models. An interesting extension of this work would be to take the predicted anomalies from a model and try to determine a smaller set of representation sequences or

root causes for them, perhaps using a similar clustering algorithm as the one used to generate pseudo-labels to cluster the anomalous sequences. Another interesting extension would be to study whether it is possible to give a human-readable summary of *why* a sequence is considered anomalous and suggest what action should be taken to correct the behaviour.

Source Criticism

Most of the sources in this work come from scientific databases such as ACM Digital Library or IEEE Xplore. For the primary sources that we relied on when making the method, such as [36, 41, 6], special care has been taken to ensure that they are peer-reviewed and on the main track for a relevant and sufficiently large conference. All papers cited in this work have been found using either Google Scholar or as references in other papers. The research in log anomaly detection using deep learning has been published recently, and it is an expanding field. However, since the field lacks a standard method of evaluation it is difficult to determine which model is the current state-of-the-art.

Furthermore, since most of the papers published on log anomaly detection are published in recent years, no or few derivations of published methods have been produced. Only papers that are well cited given the short amount of time since publication are used, this amount is difficult to determine given the size of the research field and the recency of the publications.

Finally, as far as we are aware, this is the first time cost-sensitive empirical risk minimisation methods from PU learning such as [21, 36] are used in the field of log anomaly detection. However, since PU learning is a binary classification task, and it has previously been applied in other variants of anomaly detection, applying it to log anomaly detection is justified.

5.3 The work in a wider context

This section aims to discuss the work in a wider context, such as societal, ethical and environmental factors related to the thesis.

Ethical and Societal aspects

While deploying automated log anomaly detection can contribute positively to society by helping organisations to detect anomalous behaviour early and thereby prevent or minimise damage, there are also negative aspects to it. One such negative aspect is that deploying such models can lead to the replacement of human workers, raising questions about the impact on employment and job security. Further, they heavily rely on historical data and cannot interpret the data in the same way a human can. Relying too heavily on an automated system can therefore make critical organisations such as healthcare providers susceptible to attacks and failures which would easily be avoided should a human do the task. As with any technology it is therefore essential to carefully consider the potential societal and ethical impact of automated log anomaly detection models to ensure their development and use are responsible and beneficial.

Environmental impact

As always when using deep learning methods the environmental impact must be considered. When training deep learning models large amounts of electricity are used to power the systems driving the training process. Further, usage of the models in practice requires computers to be permanently powered, looking for anomalies by using high-consumption devices such as GPUs. In this work this factor is somewhat mitigated when compared to the current state-of-the-art in deep learning by the usage of GRU models which consume a lower amount of power than the popular transformer-based architectures. Consideration of

the power consumption should still be made however when deploying such a model in practice. One way of reducing the environmental impact could be by avoiding the usage of the model during hours when the electrical grid is near maximum capacity, or using the model only infrequently, allowing the computer running it to turn off in between.

6 Conclusion

As the size of systems continues to grow the task of detecting malfunctioning systems is becoming more dependent on automated systems. This work investigates methods based on detecting anomalies in the systems based on the log files produced by borrowing methods from the field of PU learning and based on previous work in the area of log anomaly detection. Section 6.1 provides concise answers to the research questions asked and section 6.2 discusses possible extensions of this work.

6.1 Research Questions

1. *How do the word embeddings impact the f1-score of the downstream task?*

   Using FastText embeddings instead of GloVe embeddings consistently improved the performance of the baseline, but provided mixed results when combined with the other extensions, thus indicating that out-of-vocabulary tokens are not a primary factor for a low performance on the task. Further, replacing the GloVe word embeddings with FastText embeddings reduced the training time by approximately 30% for all models studied in this thesis. This suggests that using FastText embeddings is viable on certain datasets, but should not be an obvious first choice. On BGL, using FastText embeddings never decreased the performance when combined with *IR*, *nnPU* or *Imb-nnPU* and increased the performance for the baseline, thus proving to be a good choice. On HDFS and SDBS however, using FastText decreased the performance of all models except for the baseline, and is therefore not a good choice on these datasets.

2. *Which methods are appropriate for training anomaly detection models on normal and unlabelled log files, and what is the impact of them on the f1-score of the downstream task?*

   This work shows that applying two different cost-sensitive empirical risk minimisation when training a classifier significantly increased the performance on the downstream task, giving a new state-of-the-art performance on the public datasets considered as far as we are aware. Further, the reweighting scheme utilised in *Imb-nnPU* was shown to reduce the robustness, resulting in more frequent low scores. The difference in robustness between the methods is displayed in experiment 4.2 where *nnPU* showed zero results below 0.9, while the worst result for *Imb-nnPU* was below in eight out of the

twelve data splits on HDFS in experiment 4.2. Further, the average standard deviation on HDFS was 0.007 for *nnPU* and 0.14 for *Imb-nnPU*.

3. *Is the system usable in a live system in terms of effectiveness and efficiency?*

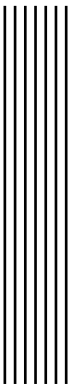
   This work found that the log anomaly detection systems studied are effective in terms of accuracy, training time and prediction time. If efficiency is crucial, it is preferable to use the smaller embeddings produced by training a FastText model. This might however reduce the performance of the model depending on the structure of the dataset. Further, disregarding the problem of several thousand raised alarms for each real system error as a delimitation, all methods studied in this work were shown to be effective at detecting anomalous sequences. *nnPU* and *Imb-nnPU* are particularly effective and were proved to be as effective as the current system in use at Sectra. Ultimately, further research is necessary to determine whether the models are truly capable of identifying anomalies, or if they are simply recognising the pre-defined regular expressions that were used to label the sequences in this study. If the latter is the case, then these methods may not be more efficient than the traditional approach of manually writing regular expressions, as regular expressions would still be required for training the models.

6.2 Future Work

In this thesis, a state-of-the-art log anomaly detection pipeline was extended by the use of new word embeddings together and an application of results produced in the field of PU learning. The result indicates that further performance improvements are possible, especially using methods for the evolving field of PU learning. However, newer research in the field of PU learning also includes other methods not used in this thesis, such as regularisation by iterative training of several networks, teacher-student networks or the semi-supervised inspired method of pseudo-labelling the unlabelled set as [38] does, using MixMatch [4]. By extending the pipeline used in this thesis with some or all of these methods additional robustness and performance gain might be possible.

Regarding the datasets, effort should be made to generate new datasets or in other ways make the problem more difficult as BGL can be considered saturated using the PU learning methods studied in this thesis. Additionally, another possible direction of future work would be to study the impact of different grouping strategies such as fixed, sliding or session windows and determine which method is the most appropriate to use. Defining a standard grouping and data-splitting method would make comparing results from different papers considerably easier and potentially enable the production of better methods.

Finally, there are questions remaining about the usefulness of this kind of anomaly detection in practice as the number of reported anomalies can be much higher than what is reasonable to handle, and the detected anomalies are already somewhat distinguishable. An interesting extension of this work would be to apply the methods to a real-world system where detected anomalies might cause a real impact, and study whether the automated systems in this study can be of use to the company when searching for anomalies.



Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] Jessa Bekker and Jesse Davis. "Learning From Positive and Unlabeled Data: A Survey". In: *CoRR* abs/1811.04820 (2018). arXiv: 1811.04820. URL: http://arxiv.org/abs/1811.04820.

[3] Jessa Bekker, Pieter Robberechts, and Jesse Davis. "Beyond the Selected Completely At Random Assumption for Learning from Positive and Unlabeled Data". In: (2018). DOI: 10.48550/ARXIV.1809.03207. URL: https://arxiv.org/abs/1809.03207.

[4] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. "Mixmatch: A holistic approach to semi-supervised learning". In: *Advances in neural information processing systems* 32 (2019).

[5] Gilles Blanchard, Gyemin Lee, and Clayton Scott. "Semi-Supervised Novelty Detection". In: *J. Mach. Learn. Res.* 11 (2010), pp. 2973–3009. ISSN: 1532-4435.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. *Enriching Word Vectors with Subword Information*. 2016. DOI: 10.48550/ARXIV.1607.04606. URL: https://arxiv.org/abs/1607.04606.

[7] Marta Catillo, Antonio Pecchia, and Umberto Villano. "AutoLog: Anomaly detection by deep autoencoding of system logs". In: *Expert Systems with Applications* 191 (2022), p. 116263.

[8] Xuxi Chen, Wuyang Chen, Tianlong Chen, Ye Yuan, Chen Gong, Kewei Chen, and Zhangyang Wang. *Self-PU: Self Boosted and Calibrated Positive-Unlabeled Training*. 2020. DOI: 10.48550/ARXIV.2006.11280. URL: https://arxiv.org/abs/2006.11280.

[9] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. "Experience report: Deep learning-based system log analysis for anomaly detection". In: *arXiv preprint arXiv:2107.05908* (2021).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[11] Emilio Dorigatti, Jann Goschenhofer, Benjamin Schubert, Mina Rezaei, and Bernd Bischl. *Positive-Unlabeled Learning with Uncertainty-aware Pseudo-label Selection*. 2022. arXiv: 2201.13192 [stat.ML].

[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning". In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, pp. 1285–1298.

[13] Marthinus Christoffel Du Plessis, Gang Niu, and Masashi Sugiyama. "Convex Formulation for Learning from Positive and Unlabeled Data". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 1386–1394.

[14] Haixuan Guo, Shuhan Yuan, and Xintao Wu. "Logbert: Log anomaly detection via bert". In: *2021 international joint conference on neural networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[15] 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)*. 2017, pp. 33–40. DOI: 10.1109/ICWS.2017.13.

[16] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. "Identifying impactful service system problems via log analysis". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2018, pp. 60–70.

[17] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. "Experience report: System log analysis for anomaly detection". In: *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE. 2016, pp. 207–218.

[18] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. "Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics". In: *CoRR* abs/2008.06448 (2020). arXiv: 2008.06448. URL: https://arxiv.org/abs/2008.06448.

[19] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. "Do We Need Zero Training Loss after Achieving Zero Training Error?" In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020.

[20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[21] Ryuichi Kiryo, Gang Niu, Marthinus C. du Plessis, and Masashi Sugiyama. *Positive-Unlabeled Learning with Non-Negative Risk Estimator*. 2017. DOI: 10.48550/ARXIV.1703.00593. URL: https://arxiv.org/abs/1703.00593.

[22] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. "Log clustering based problem identification for online service systems". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. 2016, pp. 102–111.

[23] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering". In: *The Journal of Open Source Software* 2.11 (2017), p. 205.

[24] Weibin Meng, Ying Liu, Shenglin Zhang, Federico Zaiter, Yuzhe Zhang, Yuheng Huang, Zhaoyang Yu, Yuzhi Zhang, Lei Song, Ming Zhang, et al. "Logclass: Anomalous log identification and classification with partial labels". In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1870–1884.

[25] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." In: *IJCAI*. Vol. 19. 7. 2019, pp. 4739–4745.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. DOI: 10.48550/ARXIV.1310.4546. URL: https://arxiv.org/abs/1310.4546.

[28] Adam Oliner and Jon Stearley. "What Supercomputers Say: A Study of Five System Logs". In: *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. DSN '07. USA: IEEE Computer Society, 2007, pp. 575–584. ISBN: 0769528554. DOI: 10.1109/DSN.2007.103. URL: https://doi.org/10.1109/DSN.2007.103.

[29] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[30] Marthinus Christoffel du Plessis, Gang Niu, and Masashi Sugiyama. "Class-prior Estimation for Learning from Positive and Unlabeled Data". In: *CoRR* abs/1611.01586 (2016). arXiv: 1611.01586. URL: http://arxiv.org/abs/1611.01586.

[31] PyTorch, ed. *GRU*. Feb. 14, 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html.

[32] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. "Improving language understanding by generative pre-training". In: (2018).

[33] Harish G. Ramaswamy, Clayton Scott, and Ambuj Tewari. "Mixture Proportion Estimation via Kernel Embedding of Distributions". In: *CoRR* abs/1603.02501 (2016). arXiv: 1603.02501. URL: http://arxiv.org/abs/1603.02501.

[34] Alexander Rehmer and Andreas Kroll. "On the vanishing and exploding gradient problem in Gated Recurrent Units". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 1243–1248.

[35] Hyebin Song, Bennett J. Bremer, Emily C. Hinds, Garvesh Raskutti, and Philip A. Romero. "Inferring Protein Sequence-Function Relationships with Large-Scale Positive-Unlabeled Learning". In: *Cell Systems* 12.1 (2021), 92–101.e8. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2020.10.007. URL: https://www.sciencedirect.com/science/article/pii/S2405471220304142.

[36] Guangxin Su, Weitong Chen, and Miao Xu. "Positive-Unlabeled Learning from Imbalanced Data". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 2995–3001. DOI: 10.24963/ijcai.2021/412. URL: https://doi.org/10.24963/ijcai.2021/412.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[38] Zhuowei Wang, Jing Jiang, and Guodong Long. "Positive Unlabeled Learning by Semi-Supervised Learning". In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 2976–2980. DOI: 10.1109/ICIP46576.2022.9897738.

[39] Thorsten Wittkopp, Dominik Scheinert, Philipp Wiesner, Alexander Acker, and Odej Kao. *PULL: Reactive Log Anomaly Detection Based On Iterative PU Learning*. 2023. DOI: 10.48550/ARXIV.2301.10681. URL: https://arxiv.org/abs/2301.10681.

[40] 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*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 117–132. ISBN: 9781605587523. DOI: 10.1145/1629575.1629587. URL: https://doi.org/10.1145/1629575.1629587.

[41] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. "Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021, pp. 1448–1460. DOI: 10.1109/ICSE43902.2021.00130.

[42] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. "Robust log-based anomaly detection on unstable log data". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 807–817.

[43] Yao Zhou, Jianpeng Xu, Jun Wu, Zeinab Taghavi, Evren Korpeoglu, Kannan Achan, and Jingrui He. "PURE: Positive-unlabeled recommendation with generative adversarial network". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2409–2419.

[44] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. "Tools and Benchmarks for Automated Log Parsing". In: *CoRR* abs/1811.03509 (2018). arXiv: 1811.03509. URL: http://arxiv.org/abs/1811.03509.

A Hardware

This appendix specifies the hardware of the computers used in this thesis. The hardware for computer A, B, and C can be found in table A.1, A.2, and A.3 respectively.

Table A.1: Hardware in computer A

Processor	11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz
GPU	NVIDIA RTX A2000 Laptop GPU
Dedicated GPU Memory	4 GB
RAM	32 GB

Table A.2: Hardware in computer B

Processor	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
GPU	NVIDIA T500 Laptop GPU
Dedicated GPU Memory	4 GB
RAM	32 GB

Table A.3: Hardware in computer C (Azure system Standard_NC6 [1])

Processor	Intel Xeon E5-2690 v3 (Haswell)
GPU	NVIDIA Tesla K80
Dedicated GPU Memory	12 GB
RAM	56 GB

[1] https://learn.microsoft.com/en-us/azure/virtual-machines/nc-series

B Varying known normals full tables

In this appendix, all results from the experiment explained in section 3.5 are displayed. The results for BGL are displayed in Table B.1, HDFS in Table B.2 and SDBS in Table B.3.

Table B.1: Performance for the different models with varying known normals on BGL.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.977	0.998	0.988	0.998	0.998	0.998
0.972	0.998	0.985	0.979	0.999	0.989
0.941	0.998	0.969	0.973	0.998	0.985
0.938	0.999	0.967	0.954	0.998	0.975
0.991	0.910	0.949	1.000	0.890	0.942
0.021	0.035	0.014	0.017	0.043	0.020
0.964	0.981	0.972	0.981	0.977	0.978

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.845	0.998	0.915	0.801	1.000	0.889	0.888	0.999	0.940	0.804	0.998	0.891
0.845	0.937	0.889	0.866	0.897	0.881	0.820	0.999	0.900	0.841	0.897	0.868
0.784	0.999	0.879	0.780	0.996	0.875	0.850	0.895	0.872	0.822	0.896	0.857
0.795	0.897	0.843	0.803	0.898	0.848	0.831	0.897	0.862	0.806	0.898	0.849
0.762	0.913	0.831	0.786	0.897	0.838	0.775	0.896	0.831	0.761	0.898	0.823
0.033	0.042	0.031	0.031	0.049	0.020	0.037	0.050	0.037	0.027	0.041	0.022
0.806	0.949	0.871	0.807	0.937	0.866	0.833	0.937	0.881	0.807	0.917	0.858

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
0.995	0.998	0.996	0.999	0.998	0.999
0.992	0.999	0.995	1.000	0.998	0.999
0.991	0.999	0.995	0.998	0.998	0.998
0.984	0.999	0.991	0.998	0.998	0.998
0.005	0.000	0.002	0.001	0.000	0.000
0.992	0.998	0.995	0.999	0.998	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.922	0.998	0.959	0.912	0.998	0.953	0.872	0.998	0.931	0.987	0.997	0.992
0.879	0.998	0.935	0.898	0.998	0.945	0.864	0.998	0.926	0.917	0.999	0.956
0.856	0.998	0.922	0.825	0.999	0.904	0.838	0.998	0.911	0.848	0.998	0.917
0.799	0.999	0.888	0.813	0.999	0.897	0.807	0.999	0.893	0.781	0.999	0.877
0.771	0.999	0.870	0.734	1.000	0.847	0.788	0.998	0.881	0.762	0.999	0.865
0.054	0.000	0.032	0.064	0.001	0.038	0.032	0.000	0.019	0.084	0.001	0.048
0.845	0.999	0.915	0.837	0.999	0.909	0.834	0.998	0.908	0.859	0.998	0.921

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	0.999	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.998	0.999	0.998	0.998	0.998	0.998
0.998	0.998	0.998	0.997	0.999	0.998
0.001	0.000	0.000	0.001	0.000	0.000
0.999	0.999	0.999	0.999	0.998	0.998

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.835	0.999	0.910	0.989	0.999	0.994	0.807	0.998	0.892	0.983	0.998	0.990
0.810	0.998	0.895	0.858	0.998	0.922	0.787	0.998	0.880	0.948	0.998	0.972
0.809	0.998	0.893	0.847	0.999	0.917	0.833	0.895	0.863	0.928	0.999	0.962
0.793	0.999	0.884	0.814	0.999	0.897	0.740	0.999	0.851	0.871	0.999	0.930
0.761	0.999	0.864	0.799	0.998	0.888	0.806	0.897	0.849	0.926	0.892	0.909
0.024	0.000	0.015	0.067	0.000	0.037	0.031	0.050	0.017	0.036	0.042	0.029
0.802	0.999	0.889	0.861	0.998	0.924	0.795	0.957	0.867	0.931	0.977	0.953

Table B.1: Performance for the different models with varying known normals on BGL, continued.

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.998	1.000	0.998	0.999
0.997	0.998	0.998	0.995	0.998	0.996
0.001	0.000	0.000	0.002	0.000	0.001
0.999	0.998	0.998	0.999	0.998	0.998

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.998	0.998	0.939	0.999	0.968	0.991	0.998	0.994	1.000	0.998	0.999
0.994	0.998	0.996	0.933	0.999	0.965	0.986	0.998	0.992	0.990	0.999	0.995
0.974	0.999	0.986	0.926	0.997	0.960	0.963	0.995	0.978	0.990	0.997	0.993
0.966	0.999	0.983	0.912	0.999	0.954	1.000	0.890	0.942	0.983	0.999	0.991
0.833	0.998	0.908	0.803	0.999	0.890	0.710	0.999	0.830	0.955	0.999	0.977
0.061	0.001	0.034	0.051	0.001	0.029	0.111	0.043	0.062	0.015	0.001	0.008
0.953	0.998	0.974	0.903	0.999	0.947	0.930	0.976	0.947	0.984	0.998	0.991

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.998	1.000	0.998	0.999
0.997	0.998	0.998	0.999	0.998	0.999
0.996	0.998	0.997	1.000	0.998	0.999
0.996	0.998	0.997	0.999	0.998	0.999
0.983	0.998	0.991	0.999	0.998	0.999
0.006	0.000	0.003	0.000	0.000	0.000
0.994	0.998	0.996	0.999	0.998	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.998	0.998	0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999
0.983	0.998	0.991	0.999	0.998	0.999	0.999	0.999	0.999	0.999	0.998	0.999
0.983	0.998	0.991	0.999	0.998	0.999	0.999	0.998	0.999	1.000	0.998	0.999
0.981	0.998	0.990	0.998	0.998	0.998	0.999	0.998	0.998	1.000	0.998	0.999
0.983	0.890	0.934	0.983	0.998	0.991	0.997	0.998	0.998	1.000	0.997	0.998
0.006	0.043	0.023	0.006	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000
0.985	0.977	0.981	0.996	0.998	0.997	0.999	0.998	0.999	1.000	0.998	0.999

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.998	0.999	0.998	1.000	0.998	0.999
0.998	0.998	0.998	1.000	0.998	0.999
0.001	0.000	0.000	0.000	0.000	0.000
0.999	0.998	0.999	1.000	0.998	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.999	0.998	0.999	0.999	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.998	0.998	0.998	0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999
0.985	0.998	0.992	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.998
0.984	0.998	0.991	0.996	0.998	0.997	0.998	0.998	0.998	0.999	0.997	0.998
0.007	0.000	0.004	0.001	0.000	0.001	0.001	0.000	0.000	0.000	0.000	0.000
0.993	0.998	0.996	0.999	0.998	0.998	0.999	0.998	0.999	0.999	0.998	0.999

Table B.1: Performance for the different models with varying known normals on BGL, continued.

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	0.999	0.998	0.999
1.000	0.998	0.999	0.999	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.998	0.998	0.998	1.000	0.998	0.999
0.001	0.000	0.000	0.000	0.000	0.000
0.999	0.998	0.999	1.000	0.998	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.999	0.999	1.000	0.998	0.999
0.996	0.998	0.997	0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999
0.985	0.998	0.992	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.984	0.998	0.991	0.999	0.998	0.999	0.999	0.998	0.999	1.000	0.998	0.999
0.982	0.999	0.991	0.998	0.998	0.998	0.999	0.998	0.999	1.000	0.998	0.999
0.007	0.000	0.003	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.990	0.999	0.994	0.999	0.998	0.999	0.999	0.999	0.999	1.000	0.998	0.999

(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
1.000	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999
0.000	0.000	0.000	0.000	0.000	0.000
0.999	0.998	0.999	1.000	0.998	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.999
0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.998
1.000	0.998	0.999	0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.997	0.998
0.999	0.998	0.999	1.000	0.998	0.999	1.000	0.998	0.999	1.000	0.997	0.998
0.984	0.998	0.991	0.999	0.998	0.999	1.000	0.998	0.999	0.996	0.997	0.997
0.006	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.001
0.996	0.998	0.997	0.999	0.998	0.999	1.000	0.998	0.999	0.999	0.998	0.998

Table B.2: Performance for the different models with varying known normals on HDFS.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.942	0.970	0.996	0.918	0.955
1.000	0.935	0.966	0.994	0.905	0.948
0.998	0.896	0.944	0.980	0.893	0.934
0.591	0.935	0.725	0.944	0.901	0.922
0.330	0.956	0.491	0.326	0.891	0.477
0.276	0.020	0.188	0.262	0.010	0.185
0.784	0.933	0.819	0.848	0.902	0.847

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.996	0.945	0.970	0.999	0.963	0.981	0.998	0.956	0.977	0.999	0.896	0.945
0.998	0.938	0.967	0.997	0.916	0.955	0.998	0.956	0.976	0.991	0.894	0.940
0.954	0.973	0.963	0.926	0.968	0.947	0.997	0.936	0.965	0.940	0.940	0.940
0.980	0.945	0.962	0.918	0.965	0.941	0.925	0.937	0.931	0.991	0.891	0.939
0.960	0.948	0.954	0.631	0.965	0.763	0.757	0.948	0.842	0.344	0.918	0.501
0.018	0.012	0.006	0.136	0.020	0.078	0.093	0.009	0.051	0.255	0.019	0.176
0.978	0.950	0.963	0.894	0.955	0.917	0.935	0.946	0.938	0.853	0.908	0.853

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.992	0.976	0.984	0.991	0.967	0.979
0.995	0.952	0.973	0.993	0.934	0.963
0.972	0.970	0.971	0.992	0.926	0.958
0.990	0.942	0.965	0.985	0.916	0.949
0.973	0.945	0.959	0.975	0.909	0.941
0.010	0.014	0.008	0.007	0.020	0.013
0.984	0.957	0.970	0.987	0.930	0.958

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.992	0.979	0.986	1.000	0.976	0.988	0.991	0.978	0.984	0.987	0.966	0.976
0.997	0.943	0.969	0.995	0.980	0.987	0.998	0.962	0.980	0.997	0.918	0.956
0.999	0.939	0.968	0.998	0.966	0.982	0.990	0.951	0.970	0.983	0.926	0.954
0.955	0.965	0.960	0.999	0.963	0.981	0.999	0.940	0.968	0.915	0.953	0.934
0.970	0.949	0.959	0.981	0.975	0.978	0.992	0.936	0.963	0.258	0.917	0.403
0.017	0.015	0.010	0.007	0.006	0.004	0.004	0.015	0.008	0.286	0.020	0.221
0.983	0.955	0.968	0.995	0.972	0.983	0.994	0.953	0.973	0.828	0.936	0.844

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.998	0.995	0.997	0.986	0.997	0.991
0.986	0.997	0.991	0.981	0.998	0.989
0.960	0.958	0.959	0.920	0.992	0.955
0.933	0.982	0.957	0.930	0.981	0.955
0.944	0.963	0.953	0.123	0.963	0.219
0.025	0.016	0.019	0.333	0.013	0.302
0.964	0.979	0.971	0.788	0.986	0.822

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.997	0.970	0.983	0.997	0.977	0.987	1.000	0.967	0.983	0.995	0.900	0.945
0.993	0.952	0.972	0.989	0.981	0.985	0.994	0.963	0.978	1.000	0.888	0.941
1.000	0.875	0.934	0.999	0.967	0.983	1.000	0.941	0.969	0.896	0.929	0.912
0.916	0.897	0.906	0.997	0.964	0.981	0.956	0.954	0.955	0.932	0.865	0.897
0.114	0.961	0.204	0.930	0.877	0.903	0.108	0.941	0.194	0.804	0.976	0.882
0.346	0.038	0.299	0.026	0.039	0.032	0.352	0.011	0.311	0.072	0.038	0.024
0.804	0.931	0.800	0.983	0.953	0.968	0.812	0.953	0.816	0.925	0.912	0.915

Table B.2: Performance for the different models with varying known normals on HDFS, continued.

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.869	0.992	0.926	0.951	0.997	0.973
0.828	0.977	0.896	0.910	0.972	0.940
0.797	0.985	0.881	0.922	0.958	0.940
0.347	0.854	0.494	0.929	0.945	0.937
0.095	0.978	0.174	0.213	0.878	0.343
0.310	0.052	0.296	0.286	0.040	0.242
0.587	0.957	0.674	0.785	0.950	0.827

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.733	0.859	0.791	0.095	0.971	0.173	0.954	0.927	0.940	0.985	0.767	0.862
0.789	0.783	0.786	0.088	0.903	0.160	0.603	0.903	0.723	0.083	0.833	0.150
0.665	0.924	0.773	0.088	0.911	0.160	0.089	0.923	0.163	0.077	0.778	0.139
0.091	0.933	0.166	0.082	0.854	0.149	0.082	0.828	0.149	0.069	0.890	0.127
0.077	0.775	0.141	0.080	0.820	0.146	0.069	0.880	0.128	0.063	0.793	0.116
0.318	0.067	0.309	0.005	0.052	0.010	0.360	0.036	0.343	0.365	0.045	0.292
0.471	0.855	0.531	0.087	0.892	0.158	0.359	0.892	0.421	0.255	0.812	0.279

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.997	0.998	0.996	0.997	0.997
0.999	0.997	0.998	0.997	0.997	0.997
0.999	0.996	0.997	0.957	0.997	0.976
0.999	0.996	0.997	0.924	0.998	0.959
0.999	0.995	0.997	0.849	0.998	0.917
0.000	0.001	0.000	0.055	0.000	0.030
0.999	0.996	0.997	0.944	0.997	0.969

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.997	0.998	0.999	0.996	0.997	0.998	0.996	0.997	0.996	0.997	0.996
0.998	0.997	0.997	0.999	0.994	0.997	0.998	0.996	0.997	0.994	0.997	0.996
0.999	0.995	0.997	0.997	0.996	0.997	0.999	0.995	0.997	0.995	0.995	0.995
0.989	0.997	0.993	0.995	0.998	0.996	0.999	0.993	0.996	0.993	0.997	0.995
0.998	0.986	0.992	0.988	0.997	0.993	0.998	0.992	0.995	0.988	0.998	0.993
0.004	0.004	0.002	0.004	0.001	0.002	0.001	0.002	0.001	0.003	0.001	0.001
0.997	0.994	0.995	0.996	0.996	0.996	0.998	0.994	0.996	0.993	0.997	0.995

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.931	0.964	0.995	0.930	0.962
0.999	0.930	0.964	0.995	0.930	0.962
0.999	0.930	0.963	0.999	0.911	0.953
0.999	0.930	0.963	0.999	0.881	0.936
0.999	0.930	0.963	0.999	0.878	0.935
0.000	0.001	0.000	0.002	0.023	0.012
0.999	0.930	0.963	0.998	0.906	0.949

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.931	0.964	0.998	0.930	0.963	0.999	0.930	0.963	0.997	0.913	0.953
0.999	0.930	0.963	0.998	0.930	0.963	0.998	0.930	0.963	0.999	0.904	0.949
0.999	0.926	0.961	0.999	0.929	0.963	0.999	0.930	0.963	0.998	0.880	0.936
0.978	0.932	0.955	0.999	0.927	0.962	0.999	0.923	0.959	0.998	0.880	0.935
0.998	0.881	0.936	0.980	0.915	0.947	0.999	0.880	0.936	0.994	0.881	0.934
0.008	0.020	0.011	0.007	0.006	0.006	0.000	0.019	0.011	0.002	0.014	0.008
0.995	0.920	0.956	0.995	0.926	0.959	0.999	0.919	0.957	0.997	0.892	0.941

Table B.2: Performance for the different models with varying known normals on HDFS, continued.

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.993	0.998	0.995	0.997	0.997	0.997
0.987	0.998	0.993	0.990	0.998	0.994
0.974	0.997	0.985	0.987	0.997	0.992
0.909	0.998	0.952	0.900	0.998	0.946
0.732	0.998	0.845	0.076	0.998	0.141
0.098	0.000	0.057	0.359	0.000	0.337
0.919	0.998	0.954	0.790	0.997	0.814

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.998	0.997	0.998	0.998	0.997	0.997	0.995	0.997	0.996	0.997	0.997	0.997
0.982	0.997	0.989	0.998	0.997	0.997	0.928	0.996	0.961	0.997	0.996	0.996
0.979	0.998	0.989	0.994	0.997	0.996	0.923	0.995	0.958	0.995	0.997	0.996
0.978	0.997	0.987	0.991	0.998	0.994	0.285	0.996	0.443	0.985	0.997	0.991
0.962	0.996	0.979	0.985	0.998	0.991	0.253	0.997	0.404	0.920	0.997	0.957
0.012	0.001	0.006	0.005	0.001	0.002	0.334	0.001	0.269	0.030	0.000	0.016
0.980	0.997	0.988	0.993	0.997	0.995	0.677	0.996	0.752	0.979	0.997	0.987

(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.996	0.929	0.962	0.716	0.945	0.815
0.912	0.933	0.922	0.695	0.947	0.801
0.911	0.931	0.921	0.663	0.901	0.764
0.893	0.933	0.912	0.517	0.949	0.670
0.870	0.935	0.901	0.257	0.933	0.403
0.043	0.002	0.020	0.171	0.018	0.153
0.916	0.932	0.923	0.570	0.935	0.690

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.994	0.930	0.961	0.987	0.930	0.957	0.965	0.932	0.948	0.999	0.881	0.936
0.864	0.935	0.898	0.972	0.931	0.951	0.996	0.882	0.935	0.998	0.880	0.935
0.258	0.932	0.404	0.956	0.932	0.944	0.990	0.882	0.933	0.975	0.883	0.927
0.257	0.935	0.403	0.907	0.932	0.919	0.677	0.946	0.789	0.571	0.948	0.713
0.224	0.952	0.362	0.186	0.950	0.310	0.569	0.904	0.698	0.253	0.889	0.394
0.337	0.008	0.266	0.309	0.008	0.253	0.180	0.026	0.100	0.301	0.026	0.212
0.519	0.937	0.606	0.801	0.935	0.816	0.840	0.909	0.861	0.759	0.896	0.781

Table B.3: Performance for the different models with varying known normals on SBDS.

Split A of known normals **Split B of known normals**

(a) Baseline.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.981	0.935	0.957	0.942	0.953	0.947
0.975	0.909	0.941	0.936	0.896	0.916
0.935	0.908	0.921	0.837	0.950	0.890
0.893	0.944	0.918	0.894	0.882	0.888
0.737	0.927	0.821	0.836	0.915	0.874
0.089	0.014	0.047	0.046	0.028	0.026
0.904	0.925	0.912	0.889	0.919	0.903

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.440	0.966	0.605	0.442	0.966	0.607	0.424	0.901	0.576	0.437	0.899	0.588
0.417	0.913	0.573	0.410	0.897	0.563	0.401	0.973	0.568	0.419	0.925	0.576
0.406	0.941	0.567	0.395	0.975	0.562	0.406	0.918	0.563	0.396	0.949	0.559
0.411	0.906	0.566	0.409	0.897	0.562	0.401	0.910	0.556	0.401	0.918	0.558
0.407	0.909	0.563	0.398	0.917	0.555	0.391	0.941	0.552	0.393	0.940	0.554
0.013	0.023	0.015	0.017	0.034	0.019	0.011	0.026	0.009	0.016	0.017	0.013
0.416	0.927	0.575	0.411	0.930	0.570	0.404	0.929	0.563	0.409	0.926	0.567

(b) Baseline with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.957	0.956	0.957	0.987	0.961	0.974
0.928	0.949	0.938	0.993	0.930	0.960
0.892	0.966	0.928	0.956	0.957	0.956
0.885	0.958	0.920	0.894	0.957	0.924
0.802	0.944	0.867	0.816	0.946	0.876
0.052	0.008	0.030	0.066	0.011	0.035
0.893	0.955	0.922	0.929	0.950	0.938

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.428	0.955	0.591	0.438	0.953	0.600	0.424	0.969	0.589	0.422	0.944	0.583
0.416	0.966	0.581	0.415	0.957	0.579	0.424	0.952	0.587	0.414	0.957	0.578
0.411	0.970	0.577	0.413	0.962	0.578	0.413	0.960	0.578	0.411	0.959	0.576
0.410	0.976	0.577	0.408	0.954	0.571	0.400	0.977	0.568	0.402	0.952	0.565
0.409	0.970	0.575	0.400	0.983	0.569	0.401	0.955	0.565	0.400	0.957	0.564
0.007	0.007	0.006	0.013	0.011	0.011	0.010	0.009	0.010	0.008	0.005	0.007
0.415	0.967	0.580	0.415	0.962	0.579	0.412	0.963	0.577	0.410	0.954	0.573

(c) Iterative relabelling with GloVe embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	0.999	1.000	0.999
0.998	1.000	0.999	0.999	0.999	0.999
0.957	1.000	0.978	0.999	0.998	0.998
0.879	0.993	0.933	0.990	1.000	0.995
0.831	0.989	0.903	0.999	0.986	0.992
0.067	0.004	0.038	0.004	0.005	0.003
0.933	0.996	0.962	0.997	0.996	0.997

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.879	0.996	0.934	0.873	0.977	0.922	0.876	0.992	0.930	0.970	0.987	0.978
0.840	0.937	0.886	0.828	0.932	0.877	0.824	0.943	0.879	0.746	0.980	0.847
0.732	1.000	0.845	0.755	0.966	0.848	0.799	0.946	0.867	0.770	0.931	0.843
0.705	0.897	0.790	0.617	1.000	0.763	0.894	0.818	0.854	0.739	0.910	0.815
0.615	0.916	0.736	0.712	0.776	0.743	0.647	0.997	0.785	0.680	0.979	0.802
0.095	0.042	0.070	0.089	0.080	0.068	0.087	0.065	0.047	0.099	0.031	0.063
0.754	0.949	0.838	0.757	0.930	0.830	0.808	0.939	0.863	0.781	0.957	0.857

Table B.3: Performance for the different models with varying known normals on SBDS, continued.

(d) Iterative relabelling with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.996	0.995	0.996	0.999	0.999	0.999
0.995	0.971	0.983	0.999	0.996	0.998
0.954	1.000	0.976	0.999	0.994	0.996
0.958	0.993	0.976	0.989	0.992	0.990
0.957	0.989	0.973	0.980	0.998	0.989
0.019	0.010	0.008	0.008	0.003	0.004
0.972	0.990	0.981	0.993	0.996	0.994

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.723	1.000	0.839	0.776	0.958	0.857	0.844	0.926	0.883	0.812	0.987	0.891
0.744	0.917	0.822	0.761	0.906	0.827	0.731	0.858	0.790	0.777	0.974	0.864
0.693	0.971	0.809	0.701	0.916	0.794	0.637	0.987	0.774	0.762	0.976	0.856
0.827	0.743	0.783	1.000	0.624	0.769	0.647	0.903	0.754	0.627	0.996	0.769
0.583	0.984	0.733	0.799	0.718	0.756	0.593	1.000	0.744	0.954	0.566	0.710
0.079	0.094	0.037	0.102	0.130	0.037	0.089	0.053	0.050	0.105	0.167	0.068
0.714	0.923	0.797	0.807	0.825	0.801	0.690	0.935	0.789	0.786	0.900	0.818

(e) nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
0.999	1.000	1.000	1.000	0.999	1.000
1.000	1.000	1.000	1.000	0.999	1.000
1.000	0.999	0.999	1.000	0.999	1.000
0.999	1.000	0.999	1.000	0.999	1.000
0.999	1.000	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000
0.999	1.000	0.999	1.000	0.999	1.000

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	1.000	0.999	0.999	1.000	0.999	0.999	1.000	0.999	1.000
0.999	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999	1.000	0.999	1.000
0.999	1.000	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	1.000	0.999	0.999	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.998	1.000	0.999	1.000	0.998	0.999	0.999	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.999	1.000	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999

(f) nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	0.999	1.000	0.999	0.999
0.999	0.999	0.999	0.999	0.999	0.999
1.000	0.999	0.999	0.999	0.999	0.999
0.999	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000
1.000	0.999	0.999	0.999	0.999	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.999	0.999	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	0.999	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	0.999	0.999	1.000	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	0.999	0.999	0.999	1.000	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.999	1.000	0.999	0.999	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	1.000	0.999	0.999

Table B.3: Performance for the different models with varying known normals on SBDS, continued.

(g) Imbalanced nnPU with Glove embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	1.000	0.999	1.000
1.000	1.000	1.000	1.000	0.999	1.000
1.000	1.000	1.000	1.000	0.999	0.999
0.999	1.000	0.999	0.999	0.999	0.999
1.000	0.999	0.999	0.999	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000
1.000	1.000	1.000	0.999	0.999	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	0.999	1.000	0.999	1.000	0.999	1.000	1.000	0.999	1.000
1.000	0.999	1.000	0.999	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	1.000	0.999	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	0.999	0.998	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	0.999	0.998	1.000	0.999	1.000	0.999	0.999	0.998	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.000
1.000	0.999	1.000	0.999	1.000	0.999	1.000	0.999	1.000	1.000	0.999	0.999

(h) Imbalanced nnPU with FastText embeddings.

D_1			D_2		
Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	0.999	0.999
1.000	1.000	1.000	1.000	0.999	0.999
1.000	0.999	1.000	1.000	0.999	0.999
0.999	1.000	0.999	0.999	1.000	0.999
0.000	0.000	0.000	0.000	0.000	0.000
1.000	1.000	1.000	1.000	0.999	0.999

D_3			D_4			D_5			D_6		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
1.000	1.000	1.000	0.999	1.000	1.000	1.000	0.999	1.000	1.000	0.999	0.999
1.000	0.999	1.000	0.999	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	1.000	0.998	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	0.999	1.000	0.998	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999
1.000	1.000	1.000	0.998	1.000	0.999	1.000	0.998	0.999	1.000	0.999	0.999
0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1.000	0.999	1.000	0.998	1.000	0.999	1.000	0.999	0.999	1.000	0.999	0.999

C Threshold together with PLE

To cope with mislabelled anomalies spreading a normal label to clusters which should be labelled as anomalous, a threshold can be set such that a cluster is labelled as normal if the fraction of known normal sequences in the cluster exceeds a set threshold τ,

$$\frac{|D_P \cap C_i|}{|C_i|} > \tau$$

Results from implementing this with mislabelled anomalies can be seen in Table C.1 and C.2 for BGL and HDFS respectively.

Table C.1: Introducing anomalies BGL with threshold

(a) Baseline.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.923	0.726	0.813	0.885	0.998	0.938
0.843	0.729	0.782	0.900	0.727	0.805
0.760	0.730	0.745	0.828	0.730	0.776
0.749	0.729	0.739	0.746	0.730	0.738
0.727	0.653	0.688	0.733	0.729	0.731
	A_3			A_4	
0.976	0.998	0.987	0.979	0.999	0.989
0.955	0.998	0.976	0.958	0.998	0.977
0.942	0.998	0.969	0.945	0.998	0.971
0.935	0.998	0.965	0.999	0.893	0.943
0.970	0.894	0.931	0.939	0.892	0.915

(b) FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.918	0.996	0.956	0.950	0.995	0.972
0.907	0.995	0.949	0.930	0.992	0.960
0.933	0.722	0.814	0.974	0.702	0.816
0.819	0.721	0.767	0.875	0.642	0.741
0.881	0.626	0.732	0.757	0.143	0.240
	A_3			A_4	
0.999	0.998	0.998	0.998	0.998	0.998
0.998	0.998	0.998	0.999	0.998	0.998
0.993	0.998	0.996	0.998	0.998	0.998
0.982	0.998	0.990	0.998	0.998	0.998
0.965	0.998	0.981	0.934	0.998	0.965

(c) Iterative with GloVe embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.916	0.728	0.811	0.826	0.993	0.902
0.851	0.727	0.784	0.797	0.998	0.886
0.775	0.722	0.748	0.766	0.899	0.827
0.714	0.664	0.688	0.767	0.730	0.748
0.552	0.305	0.393	0.711	0.729	0.720
	A_3			A_4	
0.995	0.998	0.997	0.995	0.998	0.996
0.955	0.998	0.976	0.997	0.932	0.963
0.999	0.891	0.942	0.911	0.998	0.953
1.000	0.890	0.942	0.999	0.891	0.942
0.999	0.891	0.942	0.996	0.890	0.940

(d) Iterative with FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.872	0.997	0.930	0.824	0.993	0.901
0.975	0.717	0.826	0.765	0.998	0.866
0.922	0.652	0.764	0.618	0.493	0.548
0.662	0.723	0.691	0.704	0.281	0.402
0.640	0.243	0.352	0.801	0.223	0.349
	A_3			A_4	
0.999	0.998	0.998	0.998	0.998	0.998
0.996	0.999	0.997	0.997	0.998	0.998
0.996	0.998	0.997	0.995	0.998	0.997
1.000	0.987	0.993	0.998	0.891	0.942
0.926	0.998	0.961	0.846	0.998	0.916

Table C.2: Introducing anomalies HDFS with threshold

(a) Baseline.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.942	0.409	0.570	0.944	0.437	0.597
0.968	0.402	0.568	0.865	0.435	0.579
0.978	0.384	0.551	0.851	0.414	0.557
0.513	0.388	0.442	0.808	0.409	0.543
0.042	0.401	0.076	0.146	0.451	0.221
	A_3			A_4	
0.999	0.966	0.983	0.999	0.996	0.998
0.996	0.958	0.977	0.996	0.996	0.996
0.959	0.963	0.961	0.997	0.994	0.996
0.900	0.960	0.929	0.458	0.995	0.627
0.874	0.959	0.915	0.304	0.997	0.466

(b) FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.996	0.447	0.617	0.999	0.430	0.601
0.862	0.429	0.573	0.999	0.408	0.579
0.994	0.367	0.536	0.896	0.422	0.574
0.949	0.373	0.535	0.805	0.395	0.530
0.161	0.378	0.226	0.789	0.388	0.520
	A_3			A_4	
1.000	0.966	0.983	0.997	0.996	0.997
0.995	0.966	0.980	0.998	0.995	0.997
0.998	0.945	0.971	0.999	0.994	0.996
0.953	0.970	0.961	0.991	0.996	0.994
0.764	0.963	0.852	0.968	0.995	0.981

(c) Iterative with GloVe embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
1.000	0.424	0.595	0.875	0.364	0.515
0.738	0.301	0.427	0.791	0.335	0.471
0.731	0.282	0.407	0.802	0.262	0.395
0.133	0.464	0.206	0.444	0.323	0.374
0.053	0.376	0.093	0.533	0.255	0.345
	A_3			A_4	
0.998	0.979	0.988	1.000	0.974	0.987
0.995	0.977	0.986	0.999	0.972	0.985
0.992	0.975	0.984	1.000	0.968	0.984
0.898	0.980	0.937	0.973	0.976	0.974
0.170	0.976	0.289	0.914	0.971	0.942

(d) Iterative with FastText embeddings.

Precision	Recall	F1	Precision	Recall	F1
	A_1			A_2	
0.270	0.270	0.270	0.775	0.307	0.440
1.000	0.111	0.199	0.916	0.278	0.426
0.040	0.391	0.072	0.561	0.245	0.341
0.038	0.469	0.070	0.284	0.319	0.301
0.025	0.510	0.047	0.100	0.288	0.148
	A_3			A_4	
0.991	0.905	0.946	0.090	0.910	0.164
0.090	0.925	0.164	0.090	0.925	0.164
0.088	0.912	0.160	0.089	0.925	0.162
0.084	0.876	0.154	0.089	0.923	0.162
0.081	0.836	0.148	0.086	0.923	0.157

D Hyperparameters

This appendix describes the common hyperparameters used in the experiments. All parameters are listed in Table D.1. For DRAIN config files, we refer to the source code available at https://github.com/tigmo/camel-config.

Table D.1: Hyperparameters

General parameters	
Batch size	1024
Number of epochs	15
Dropout percentage for sequences	10 %
Surrogate loss function	BCEloss
Attention	
Attention type	MultiHead Attention [37]
Number of Attention heads	5
RNN parameters	
Bidirectional	Yes
Number of stacked RNN layers	2
Hidden dim for RNN	100
Dense layers	
NOTE: No hidden layer was used in the baseline.	
Hidden layer dimension	30 when using IR and 100 for nnPU and Imb-nnPU
Activation function	tanh