Log Message Anomaly Detection using Positive and Unlabeled Learning

by

Fatemeh Seifishahpar
B.Sc., Bu-Ali Sina University, 2003
M.Sc., Institute for Advanced Studies in Basic Sciences, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Log Message Anomaly Detection using Positive and Unlabeled Learning

by

Fatemeh Seifishahpar
B.Sc., Bu-Ali Sina University, 2003
M.Sc., Institute for Advanced Studies in Basic Sciences, 2009

Supervisory Committee

Dr. T. Aaron Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Kin Fun Li, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Log messages are widely used in cloud servers and software systems. Anomaly detection of log messages is important as millions of logs are generated each day. However, besides having a complex and unstructured form, log messages are large unlabeled datasets which makes classification very difficult. In this thesis, a log message anomaly detection technique is proposed which employs Positive and Unlabeled Learning (PU Learning) to detect anomalies. Aggregated reliable negative logs are selected using the Isolation Forest, PU Learning and Random Forest algorithms. Then, anomaly detection is conducted using a deep learning Long Short-Term Memory (LSTM) network. The proposed model is evaluated using the commonly employed Openstack, BGL, and Thunderbird datasets and the results obtained indicate that the proposed model performs better than several well-known approaches in the literature.

Contents

List of Tables

List of Figures

List of Acronyms

BGL	BlueGene/L
BLSTM	Bidirectional Long-Short Term Memory
DT	Decision Tree
DL	Deep Learning
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
IF	Isolation Forest
IKNN	Improved K-Nearest Neighbor
LSTM	Long Short-Term Memory
ML	Machine Learning
NB	Naive Bayes
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PU	Positive and Unlabeled
RF	Random Forest
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network
OC-SVM	One-Class Support Vector Machine
OOB	Out-of-Bag
SVM	Support Vector Machine
tanh	hyperbolic tangent
TF-IDF	Term Frequency-Inverse Document Frequency

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my supervisor Dr. T. Aaron Gulliver. Thank you for all the help from the beginning. Further, thanks to my committee members, Dr. Kin Fun Li and Dr. Alex Thomo, for their invaluable guidance and insightful feedback.

I also want to thank Janice Closson, Ashleigh Carlsen and Dan Mai who always helped me.

I would like to also thank my mom, dad, and siblings.

To my mom and dad!

Chapter 1

Log Message Data and Log Anomaly Detection

Modern IT services are getting more complex, and as a result, their operation and maintenance poses new challenges for personnel. On the other hand, organizations and clients demand constant, uninterrupted access to their cloud and software platforms, and an interruption can lead to serious consequences. Therefore, substantial resources have been dedicated to ensuring the reliability and accessibility of these services. One method employed to achieve this goal is logging, which refers to the process of recording events, activities, or messages from computer programs to a log file to track the behavior of applications, monitor performance, and gain insights into the program execution flow.

In this chapter, first log message data and their composing elements and structure along with three common log message datasets are introduced. Then, log message anomaly detection and related algorithms and proposed strategies in the literature are discussed. An overview of positive and unlabeled data classification is also given. Finally, the required background regarding the implemented algorithms including Isolation Forest (IF), Positive and Unlabeled Learning (PU Learning), and Random Forest (RF) along with the deep neural network in the proposed model is given.

1.1 Log Message Data

Log messages are semi-structured text messages pertaining to the system current state. Log messages typically include the following information.

Negative log message	1117838571 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.51.293532 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
Positive log message	1118958635 2005.06.16 R30-M1-NJ18-U01 2005-06-16-14.50.35.784189 R30-M1-N4-I:J18-U01 RAS APP FATAL ciod: LOGIN chdir (/p/gb2/draeger/benchmark/dat32k_061605) failed: No such file or directory

Figure 1.1: Examples of negative and positive log messages from the BGL dataset.

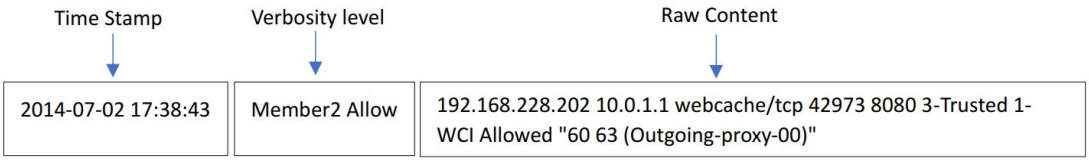

Figure 1.2: An example of a log message consisting of time stamp, verbosity level and raw content.

- Time stamp: The date and time when the event occurred.
- Verbosity level: The severity or importance of the event, e.g., INFO, ERROR.
- Source or module: The component of the system that generated the log message.
- Message content: Details about the event, action, or error.

An example of a log message consisting of these components is given in Figure 1.2. Log messages are instrumental in diagnosing issues, tracking system behavior, and providing a historical record of events. They are commonly used by developers, system administrators, and support personnel to understand what happened within a system, identify potential problems, and take appropriate action. For example, in a web server log, there may be entries related to incoming requests, responses, errors, and other relevant information. In a software application log, you might find messages related to user actions, data processing, and error conditions.

Since log messages are unstructured, they are usually parsed before being input to a Machine Learning (ML) model. Log parsing is the process of converting log data into a common format to make them machine-readable. There are several log parsing methods. However, as systems are continually developing, creating effective automated log parsing techniques is becoming more challenging [43].

There are two types of log messages. The first type includes log messages indicating successful events, actions, or operations, which are generated when a process or task completes without errors. The second type includes messages indicating errors, failures, or unsuccessful events within a system. In this thesis, the first type of messages are called negative and the second type are called positive. The reason for this

comes from the fact that in detecting anomalies in a system, finding messages indicating errors is the goal, so messages representing this state of the system are usually called positive log messages. Figure 1.1 shows examples of positive and negative logs from BlueGene/L (BGL)[1], where the positive (abnormal) log is due to a login failure in accessing a directory in the system.

Millions of logs are generated each day which makes them important for anomaly detection. However, their large and complex structure makes this task difficult [3]. Therefore automated approaches for log message analysis are needed for robust and accurate anomaly detection in a timely manner. This issue has been addressed in the literature for text-based problems, and more specifically for log anomaly detection. These methods first conduct feature extraction, and then employ ML algorithms to classify the log messages. Some approaches consider only one characteristic such as verbosity or time stamps which restricts their ability to detect anomalies.

1.2 Log Message Anomaly Detection

ML can be described as the process of learning from existing data and developing predictive solutions for unseen data [4]. In the modern world, big data and ML provide many opportunities for knowledge extraction. ML algorithms use data to identify patterns, make connections, and learn representative models. In supervised learning, the training model requires labeled data which is typically obtained manually. While enormous quantities of data are accumulated in the modern world, labeling these can be prohibitively expensive and time-consuming.

In many real-world learning problems, unlabeled and positive data are widely available while negative data can be difficult and expensive to obtain [33], [34]. For example consider the automatic diagnosis of diseases: unlabeled data are easy to collect (all patients in the database), and also positive data are readily available (the patients who have the disease), but negative data are expensive if tests for the disease are expensive since all the patients in the dataset cannot be assumed to be negative instances if they have not been tested. Thus an important problem is how to efficiently learn from unlabeled data.

Several unsupervised learning techniques have been proposed in the literature to handle unlabeled data. In [22], the IF algorithm was proposed to isolate abnormal

[1] `https://github.com/logpai/loghub/tree/master/BGL`

instances to detect anomalies. In contrast to the IF algorithm, which uses attributes and values, the extended IF algorithm proposed in [24] makes use of slopes and intercepts, which improves the performance of the IF algorithm in dealing with large-scale and complex data [28]. Support Vector Machine (SVM) [29] and One-Class SVM (OC-SVM) are two techniques developed to detect anomalies in automotive control networks [30].

An unsupervised model for log message anomaly detection was proposed in [31] which employs the IF algorithm for positive instance prediction and two deep Autoencoder networks for classification. Another approach using the Deep Learning (DL) methods Auto-LSTM, Auto-BLSTM and Auto-GRU was developed in [66] for anomaly detection and log classification.

The technique in [62] employs pruning of positive and negative logs. Reliable positive log messages are selected using a Gaussian mixture model and reliable negative logs are selected using the K-means, Gaussian mixture model and Dirichlet process Gaussian mixture model methods iteratively.

1.3 Positive and Unlabeled Data and Classification

In traditional binary classification tasks, instances are required to be labeled as either positive or negative . However, in many situations, only positive labels are identifiable while negative ones are not. For instance, recognizing customers who are interested in one product from the customer profiles. Customers who bought the product can be considered as positive instances, while the others have unknown interest.

In molecular biology [21], a number of datasets contain proteins that are known to have particular functions. In these datasets, under some restrictions, instances can only be labeled when the functions are active and observed. However, this does not mean that the unlabeled instances do not have any function. These situations are typical in many areas where we can only label positive instances and leave the others without labels. Thus, there is a huge amount of positive and unlabeled data in real-world applications. Some of these applications employ PU Learning such as text classification.

Other problems such as drug identification, disease gene, and molecular classification are still in their infancy. In addition, there are problems that have never

or only rarely been investigated using PU Learning such as sound, image and video classification and various security applications.

Semi-supervised techniques have been proposed to address the problem where labeled data is not available for training. Many real-world classification problems inherently have a small number of known positive data, no known negative data, and a large quantity of unlabeled data. This situation naturally arises in applications such as medical diagnosis or personalized advertising and is known as PU Learning. In PU Learning, a model learns from a few positive instances and unlabeled data. The assumption is that each unlabeled instance could belong to either the positive or negative class [32].

Most solutions to the PU Learning problem assume that at least a portion of the positive class is separable from the negative class. This is illustrated in Figure 1.3 where the red crosses represent positive instances, sometimes referred to as the positive subdomain or anchor set [19], and the black dots represent the unlabeled dataset. Figure 1.4 shows the standard classification boundary between positive and negative instances and the classification boundary between positive and unlabeled data from PU Learning classification algorithms.

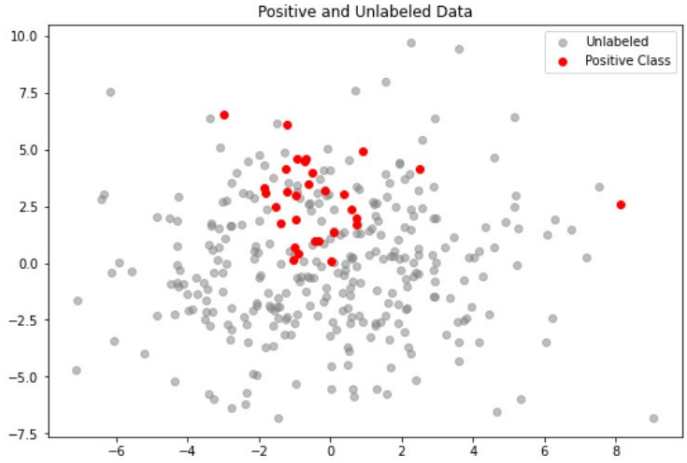

Figure 1.3: An example of positive and unlabeled data which could belong to either the positive or negative class [8].

Many papers have been published on PU Learning in the past twenty years. One approach is a two-step algorithm that first finds probable negative instances from the unlabeled data, and then estimate a classifier from the positive, unlabeled and likely negative instances [32], [37], [39]. Other approaches to overcome the lack of negative instances include disregarding unlabeled instances during training and simply learning

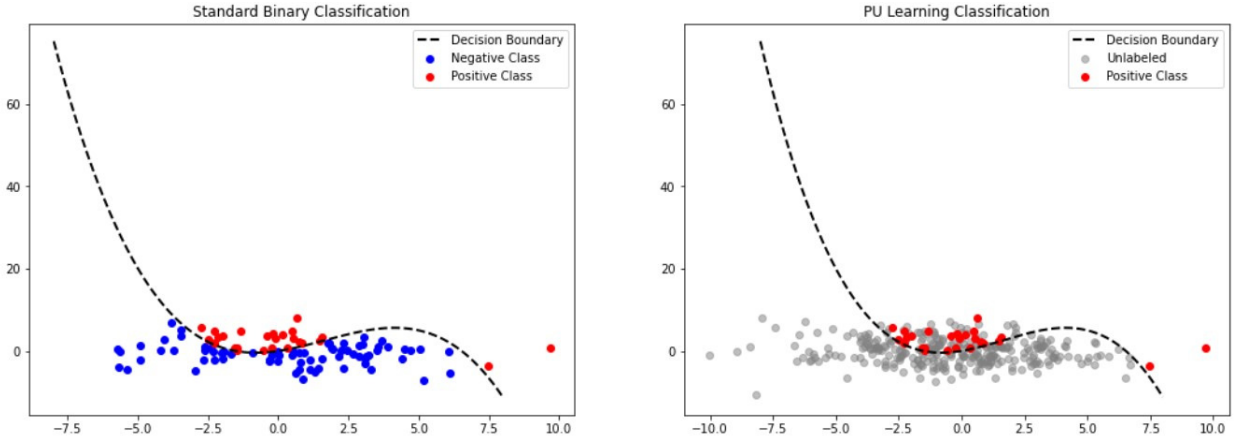

Figure 1.4: Illustration of the difference between standard binary classification and PU Learning classification [8].

from the positive instances [35] or using more advanced learning methods such as OC-SVM [36]. Another approach assigns weights to the unlabeled instances and then trains a classifier using these weighted instances [10], [40].

Inductive PU Learning involves learning to distinguish between positive and unlabeled instances, potentially by adjusting the misclassification costs for the two categories [40], [42]. Some algorithms treat unlabeled data as noisy or uncertain negatives [14], [15], or use the strategies described above, but include methods to minimize overfitting and stabilize the algorithms [16], and implement deep learning techniques such as Generative Adversarial Networks (GANs) [18]. In [13], the LogClass framework was proposed to identify and classify anomalous logs for networks and services based on partial labels. LogClass combines a word representation method, a PU Learning model, and an ML classifier.

In [17], a method was proposed for reactive anomaly detection based on estimated failure time windows provided by monitoring systems instead of labeled data. Systems often produce significantly more logs when an error occurs, therefore placing them in close temporal proximity makes sense. The proposed model uses an objective function that accounts for imbalanced data and applies an iterative learning strategy for PU Learning to identify anomalous logs.

This thesis presents an anomaly detection algorithm using ensemble learning to improve the prediction results. Ensemble learning is an ML technique that combines multiple individual models, often called base learners or weak learners to create a

more powerful and accurate prediction model. The idea behind ensemble learning is that by combining the predictions of multiple models, the overall performance can be improved, the weaknesses of individual models can be mitigated and overfitting can be reduced [44], [46].

The proposed model employs three different learners, namely Isolation Forest, PU Learning and Random Forest, to obtain reliable negative instances from unlabeled instances. Then, these reliable negative logs along with the existing positive instances are used to train a Long Short-term Memory (LSTM) network for anomaly detection. The proposed method exploits the presence of a large number of negative instances compared to positive ones in the unlabeled instances to obtain negative instances.

The proposed method leverages the strengths of the algorithms in distinguishing positive logs from other instances to acquire reliable negative instances from unlabeled instances for training. It is evaluated using the precision, recall and F-measure metrics [47], and three log message datasets, namely BGL, Openstack[2] and Thunderbird[3], are considered for evaluation. The parameters of the proposed model are the same for all datasets to evaluate the robustness of this approach. The main contributions of this research are as follows.

- A text preprocessing method is employed to prepare log message data for training.
- The prediction results from three base learners are used to identify reliable negative instances from unlabeled instances.
- An LSTM network is trained with the known positive instances and reliable negative instances for anomaly detection of unseen data.

1.4 Background

This section presents the IF, PU Learning and RF algorithms and the LSTM architecture.

[2] `https://github.com/logpai/loghub/tree/master/OpenStack`
[3] `https://github.com/logpai/loghub/tree/master/Thunderbird`

1.4.1 Isolation Forest

Isolation Forest (IF) [22] is an ensemble approach which has been used to detect anomalies. Isolation is the process of separating unusual patterns from the dataset. IF employs binary trees to detect anomalies, resulting in a linear time complexity algorithm that is well-suited for processing large datasets. The algorithm chooses data with randomly selected features. This continues until the instances are isolated. When an instance travels deeper into the tree, it means that it is less likely to be an anomaly as it requires more cuts to be isolated. Similarly, the instances which end up in shorter branches are more likely to be anomalies as the tree found them easier to distinguish from the other instances. Thus it is expected that anomalies (abnormal instances) will have a smaller average path length than normal instances [48].

The anomaly score of instance x for the IF algorithm is

$$s(x, N) = 2^{\frac{-E(h(x))}{c(N)}}, \tag{1.1}$$

where N is the number of instances trained in each tree of the forest, $E(h(x))$ is the average length of the paths in all trees, and $c(N)$ is the normalization factor defined as

$$c(N) = \begin{cases} 2H(N-1) - 2(N-1)/N & \text{if } N > 2, \\ 1 & \text{if } N = 2, \\ 0 & \text{otherwise,} \end{cases} \tag{1.2}$$

where $H(N-1)$ is the harmonic number given by $\ln(N-1) + 0.5772156649$ [49]. When an instance is at a leaf deep in the tree, the score will be close to zero, while if it is shallow the score will be close to one.

1.4.2 PU Learning

The goal of binary classification is to learn a model that can discriminate between positive and negative instances. Learning algorithms need to be provided with training data. Typically, this data contains both positive and negative instances and all data is labeled. PU Learning is a variant of this approach where the training data consists of positive and unlabeled instances. The assumption is that each unlabeled instance could belong to either the positive or negative class. There has been a surge of interest in this approach in recent years [32], [33], [52], [53].

PU Learning is a two-step technique which identifies likely negative data instances from the unlabeled data. Using these estimated negative instances and the known positive instances reduces the problem to a supervised or semi-supervised learning problem which can be solved using a variety of methods.

In its simplest form, a traditional binary classifier can be thought of as a function that provides an estimate of the conditional posterior probability that an instance belongs to the positive class given its feature values. Denote an instance feature vector as $x \in \mathbb{R}^n$. The class label $y \in \{0, 1\}$ is used to label the instance as positive ($y = 1$) or negative ($y = 0$). Using this notation, a classifier can be written as a function $f : \mathbb{R}^n \to \{0, 1\}$

$$f(x) = p(y = 1|x). \tag{1.3}$$

This posterior probability $p(y = 1|x)$ is related to the prior probability $p(y = 1)$, the likelihood $p(x|y = 1)$, and the marginal probability of observing x, $p(x)$, via Bayes' theorem given by

$$p(y = 1|x) = \frac{p(x|y = 1).p(y = 1)}{p(x)}. \tag{1.4}$$

The marginal probability $p(x)$ is typically ignored as a constant.

In PU Learning, much of the data is unlabeled and so the class label y is not available for most of the data. To deal with this, a new variable $s \in \{0, 1\}$ is introduced [40] which represents whether x is labeled ($s = 1$) or not ($s = 0$). A PU dataset is then a set of triplets (x, y, s) with x a vector of features, y the class label, and s a binary variable representing whether it is labeled.

The training set (x, y, s) consists of labeled ($s = 1$), and unlabeled ($s = 0$) sets. The likelihood that an unlabeled instance x belongs to the positive set is assigned to each unlabeled instance as a real value such that $f(x) = p(y = 1|x)$ can be estimated accurately [10], [40]. By approximating the values of $g(x) = p(s = 1|x)$ and $p(s = 1|y = 1)$, the traditional classifier $f(x)$ can be obtained from $g(x)$ as

$$f(x) = g(x)/p(s = 1|y = 1), \tag{1.5}$$

which comes from

$$\begin{aligned} p(s=1|x) &= p(y=1 \wedge s=1|x) \\ &= p(y=1|x)p(s=1|y=1,x) \\ &= p(y=1|x)p(s=1|y=1). \end{aligned} \tag{1.6}$$

Therefore, the conditional probabilities produced by a model trained on the positive and unlabeled instances differ by only a constant c from the conditional probabilities produced by training the model on only labeled instances.

In [40], three different approaches were proposed to estimate the value of $c = p(s = 1|y = 1)$. The value of $c = p(s = 1|y = 1)$ can be estimated using a trained classifier g and a validation set. Let V be the validation set drawn from the overall distribution $p(x, y, s)$ and P be the subset of instances in V that are labeled (and hence positive). A simple yet effective estimator of c is $c = \frac{1}{n}\sum_{x\in P} g(x)$, where n is the cardinality of P [40]. Therefore, the traditional binary classifier of $f(x)$ is transformed to a PU Learning classifier that needs to learn a function $g(x)$ such that $g(x) = p(s = 1|x)$ approximately.

Decision Tree (DT) is an effective algorithm that can be used as the underlying classifier $g(x)$. Decision trees are used for classification tasks and regression tasks as an approximation function. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Mechanisms such as pruning, setting the minimum number of instances required at a leaf node, and setting the maximum depth of the tree are necessary to avoid overfitting in decision trees.

The proposed model benefits from aggregating the decision tree classifier predictions which are individually trained to discriminate positive instances P from a small random subset of U. The output of each classifier assigns a prediction score to the instances. The final aggregated classifier is simply defined as the average score of the individual classifiers.

1.4.3 Random Forest

RF [54] is a supervised ML algorithm that is widely used for classification and regression problems. It builds decision trees on different instances and takes their majority vote for classification tasks. A subset of data points and a subset of features are

selected for constructing each decision tree. Each model is therefore created from bootstrap instances provided by the original data with replacement. The final decision is made based on majority voting after combining the results of all models. The strength of this method is in capturing commonalities of weak predictors (the trees) and combining them to create a powerful predictor.

The feature with the smallest entropy is selected as the root of the tree. Entropy measures the impurity of the instance values in a dataset and is given by

$$H(D) = -\sum_{i \in C} p_i \log(p_i), \tag{1.7}$$

where D represents the corresponding dataset, and C is the set of class labels. Information gain represents the difference in entropy before and after splitting based on a given attribute, and is represented by

$$I(D, a) = H(D) - \sum_{v \in cat(a)} \frac{|D_v|}{|D|} H(D_v), \tag{1.8}$$

where a represents a feature, $cat(a)$ represents the set of existing categories of v in the feature a, and D_v is the cardinality of category v in a. The feature with the highest information gain will produce the best split as it should be the best at classifying data into homogeneous subsets in the root of the tree.

1.4.4 LSTM Architecture

An LSTM network [55] is a Recurrent Neural Network (RNN) which uses cells to preserve sequence information and recall long-term dependencies [56]. LSTM networks are suitable for sequential data such as log messages [58] and are effective with big and complex data [59]. Cells are connected recurrently, meaning there are feedback connections within the network, allowing information to persist and be passed from one time step to the next. This structure is beneficial in solving the vanishing gradient problem.

Each LSTM block includes input, forget, and output gates. These gates can improve the performance by storing information longer than feed-forward neural networks [56]. A block of an LSTM network contains recurrently connected cells as shown in Figure 1.5. The input of a cell at time t is denoted by x_t and the values of the input, forget, and output gates are

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{1.9}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{1.10}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{1.11}$$

respectively, where h_{t-1} is the previous block output, b is the bias vector, W and U are weight matrices, and σ is the activation function (usually the sigmoid function). The block input at time t is given by

$$\widehat{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C), \tag{1.12}$$

where b_C is the bias vector, W_C and U_C are the weight matrices, and tanh denotes the hyperbolic tangent activation function. The cell state at time t is

$$C_t = f_t \odot C_{t-1} + i_t \odot \widehat{C}_t, \tag{1.13}$$

where $\odot$ denotes element-wise multiplication. The block output at time t is

$$h_t = o_t \odot \tanh(C_t). \tag{1.14}$$

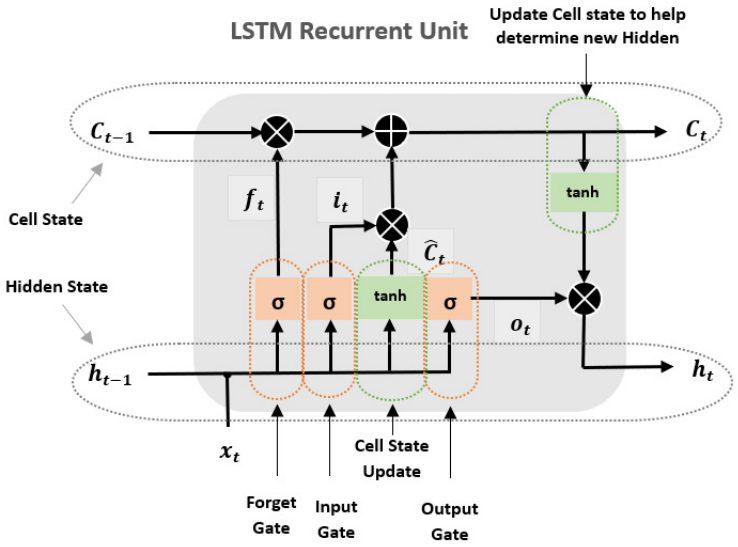

Figure 1.5: A block of an LSTM network with input, output and forget gates.

The cells remember values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cells. Forget gates decide what information to discard from a previous state by assigning a previous state, compared to a current input, a value between 0 and 1. A (rounded) value of 1 means keep the information while a value of 0 means discard it. Input gates decide which new information to store in the current state using the same system as forget gates. Output gates control which information in the current state to output by assigning a value from 0 to 1 considering the previous and current states. Selectively outputting information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions in both the current and future time steps.

The next chapter presents the classifiers and the deep neural network to build a model for classifying large-scale log message datasets.

Chapter 2

Anomaly Detection using Positive and Unlabeled Data

In this chapter, a model is proposed to extract reliable negative log messages from imbalanced log data using the IF, PU Learning, and RF algorithms. The resulting data is then used for anomaly detection with an LSTM network. The proposed model is evaluated using three labeled log message datasets, namely BGL, Openstack, and Thunderbird. Results are presented which show that the proposed model can successfully recognize negative data logs among unlabeled data to provide the LSTM network with labeled positive and negative data for training.

The rest of the chapter is organized as follows. In Section 2.1 the proposed model is described, including preprocessing the log messages, model setup, reliable negative log data extraction, and model training and anomaly detection. The experimental results are given in Section 2.3, and Section 2.4 provides a discussion of the model performance as well as the model settings and parameters.

2.1 Proposed Model

The proposed model has two steps through which the model learns from a few positive instances and a set of unlabeled data. In the first step, three different classifiers are employed to obtain reliable negative logs which are those that all classifiers agree are negative. In the second step, anomalies are detected in unseen data using an LSTM network trained with the known positive instances and the reliable negative instances from the first step of the algorithm.

2.1.1 Preprocessing and Model Setup

Before the first step, text preprocessing including changing letters to lowercase and removing hyphens is performed. Then, tokanization, an initial transformation method from Natural Language Processing (NLP), is applied to enable the content of the log messages to be input to an algorithm. We refer to the smallest units of meaning in a log message as tokens, which can be words, phrases, or even individual characters. For example, the log message

> "[INFO] User authentication successful for user john."

is converted to the vector

> "[‘[’, ‘INFO’, ‘]’, ‘User’, ‘authentication’, ‘successful’, ‘for’, ‘user’, ‘john’, ‘.’]"

using the Natural Language Toolkit (NLTK) library for tokenization in Python. Consequently, each log can be interpreted as a sequence of tokens.

In this work, sentences were truncated to 40 tokens, and sentences including less than five tokens were removed. Then, the number of appearances of each token in the dataset was counted and tokens were ordered from most frequent to least frequent. Each token was given an index starting from zero and the indices were used as the dataset features.

The training set for PU Learning was obtained as follows. First, a small subset of positive (anomalous) instances were randomly selected from available positive instances in the dataset. As shown in Figure 2.1 and Table 2.1, the size of this subset was determined by taking different numbers of instances and choosing the smallest number of positive instances required to have a well-trained LSTM network. The original labels of the remaining positive instances along with the negative instances were first stored as the ground truth labels and then their labels were changed to zero to create an unlabeled dataset, so the assumption of a PU dataset (where each unlabeled instance could be belong to either the positive or negative class), is met. Finally, the data was shuffled and split with 5% for training and 95% for testing for the Openstack dataset, and 1% for training set and 99% for testing for the BGL and Thunderbird datasets. The unlabeled training sets are small to keep the computational complexity low.

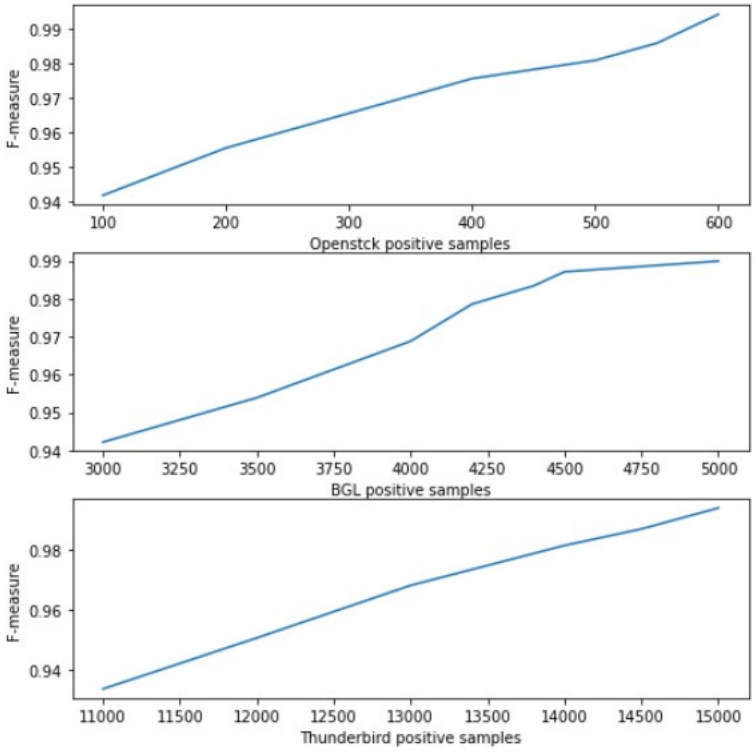

Figure 2.1: F-measure versus the number of positive instances from the Openstack, BGL and Thunderbird datasets used to train the LSTM network.

Table 2.1: The original number of positive and negative instances along with the number of positive and unlabeled instances used in PU Learning.

Dataset	Positive	Negative	PU Positive set	PU Unlabel set
OpenStack	18,434	137,074	600 (0.03%)	7889
BGL	348,460	4,399,502	5000 (0.01%)	17,174
ThunderBird	3,000,000	3,248,239	15000 (0.05%)	62,332

2.2 Reliable Negative Instance Extraction

The proposed approach includes first extraction of reliable negative instances and then using these negative instances along with the existing positive instances to train an LSTM network in order to detect anomalies in log message data. In this section, we discuss the first phase of the proposed method.

2.2.1 Isolation Forest Model Parameters

Considering different algorithms and tuning their parameters based on the nature of the data is often necessary to obtain good performance. The IF algorithm is used to detect anomalies by isolating them rather than identifying normal instances. The key idea is that anomalies are likely to be few in number and more isolated in the feature space. The IF algorithm uses a collection of isolation trees to separate anomalies. Each tree is constructed by recursively partitioning the data until each instance is isolated in a leaf node. At each split in the tree, a random feature is selected, and a random split within the range of the selected feature is chosen. This randomness contributes to the efficiency of the algorithm.

The IF algorithm is compatible with high-dimensional datasets and is scalable to large datasets, so it is a suitable choice for anomaly detection. It is resilient to overfitting because each tree in the ensemble focuses on isolating instances in a sparse, random manner.

The contamination rate is the proportion of anomalies in the dataset and varies between 0 and 0.5. In PU Learning, the assumption is that very few positive instances are available, so the contamination rate should be a small number. Another parameter that needs to be tuned is the size of the class estimator. Figure 2.2 suggests that based on the F-measure for different numbers of estimators, 900 trees is a good choice. The choice of parameters often involves a tradeoff between model accuracy and computational efficiency. Figure 2.2 shows that increasing the number of estimators to more than 900 trees has a minimal effect on the F-measure and increases the computation time. The rest of the parameters were set to the default values.

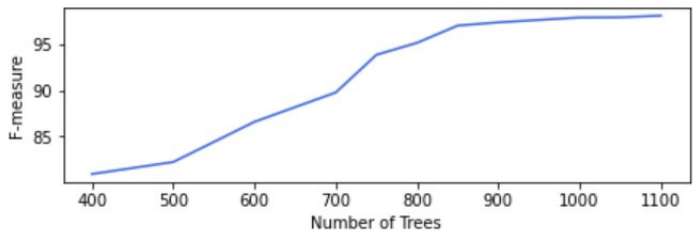

Figure 2.2: F-measure versus the number of estimators employed in the IF algorithm with the BGL dataset.

2.2.2 PU Learning Model Parameters

Most traditional binary classifiers are unable to learn without negative labels. PU Learning chooses the most suitable probability estimator $g(x)$ to label the unlabel data [9], [32]. A classifier trained using this data is expected to perform better than the same classifier trained with unlabeled data.

In this work, the Naive Bayes (NB), OC-SVM and DT algorithms are considered as the underlying probability estimators in the PU Learning algorithm to investigate their impact on model prediction using the BGL log message dataset. NB is a probabilistic ML algorithm based on Bayes' theorem. It often performs well with high-dimensional data such as in text classification and spam filtering [26], [27]. OC-SVM, on the other hand, aims to distinguish one class of data from the other given only positive labeled data [11], [12]. It differentiates instances of one class by learning from single class instances during training. The DT algorithm constructs a tree-like structure from the data by splitting the training dataset into disjoint subsets at each node based on the feature that yields the highest information gain (or reduction in entropy) [25].

Table 2.2 presents the average performance results for 10 runs with the NB, OC-SVM and DT algorithms along with the average running times. These results indicate that the DT algorithm outperforms the other estimators and is more efficient. Further, the NB and OC-SVM algorithms performed poorly even with parameter tuning. Although the NB algorithm has a reasonable running time compared to the OC-SVM algorithm, the performance is much worse. One reason may be related to the fact that the NB algorithm assumes that features follow a certain distribution (usually Gaussian or multinomial). If the actual distribution deviates significantly from the assumption, performance can be affected. The OC-SVM algorithm also suffers from a lack of information about the distribution of negative data among the unlabeled data, and requires a large amount of positive training data to obtain an accurate boundary [57].

Note that in bagging, the data is randomly sampled to create subsets with replacement allowed, meaning that the same instance can appear in multiple subsets. After the subsets have been classified, the final output class is selected by voting over the bagged labels. In bagging, only unlabeled data was randomly sampled so the known positive instances were not included. This is due to the limited amount of labeled positive data available.

Table 2.2: Average performance for 10 runs of the model with different estimators for PU Learning.

Estimator	Dataset	Label	Precision	Recall	F-measure	Time (s)
Naive Bayes	**BGL**	0	77.4%	78.6%	77.9%	1084
		1	75.1%	75.9%	75.4%	
	Openstack	0	75.4%	77.6%	76.4%	865
		1	71.1%	70.5%	70.8%	
	Thunderbird	0	73.7%	69.2%	71.3%	1255
		1	70.9%	71.3%	71.1%	
One-class SVM	**BGL**	0	84.5%	83.7%	84.1%	1720
		1	85.9%	85.3%	85.6%	
	Openstack	0	84.9%	83.3%	84.6%	1339
		1	79.1%	81.8%	80.4%	
	Thunderbird	0	81.1%	80.3%	80.7%	1802
		1	79.9%	78.5%	79.2%	
Decision Tree	**BGL**	0	89.9%	93.3%	91.5%	1415
		1	88.1%	91.8%	89.9%	
	Openstack	0	92.1%	92.3%	92.1%	989
		1	88.1%	91.8%	89.9%	
	Thunderbird	0	89.9%	90.2%	90.1%	1022
		1	90.4%	88.8%	89.6%	

2.2.3 Random Forest

Random Forest is a versatile ML algorithm that is effective in classification tasks. One of the most important characteristics of the RF algorithm is that the diversity of trees in the ensemble helps capture different aspects of the data distribution, which offers improved detection performance compared to other algorithms. It can also capture complex relationships in the data. An advantage of using the RF algorithm for anomaly detection is that it does not assume a specific data distribution. This is important as anomalies often do not follow the majority of the patterns [22].

In the RF algorithm, the Out-of-Bag (OOB) score is used for model evaluation. This score measures the performance of the model on the training data that was not used during tree construction. This score can be employed to assess the ability of the model to detect anomalies. After training, each tree is tested on the OOB instances that were not part of its training set. The predictions on these instances are then aggregated and used to identify potential anomalies. If an instance receives consistently low OOB scores or deviates from the majority predictions across the trees, it is flagged as an anomaly.

Tuning the parameters of the RF algorithm, such as the number of trees, the depth of each tree, and the minimum instances per leaf, can impact the performance of the algorithm. The same number of trees was used for the RF algorithm as the IF algorithm (900) and the rest of the parameters were set to the default values.

2.2.4 Negative Instance Extraction

After investigating different algorithms and tuning the corresponding parameters, the negative instances were identified as follows. In the first step, the IF classifier was employed to classify the PU dataset containing a small number of positive instances and a set of unlabeled instances (with different ratios for the Openstack, BGL, and Thunderbird datasets according to Table 2.1), and the predicted positive and negative labels were obtained.

In the second step, PU Learning was employed to classify the set of positive and unlabeled instances. Bagging [5] was employed to improve the prediction results. In bagging, random groups of instances from the training set are selected with replacement. Then, the model is trained separately on each group and a majority vote of the predictions is chosen as the result. The DT algorithm [25] was used as the underlying estimator.

The third step included implementing the RF algorithm to classify and then collecting the labels that the algorithm predicted from the set of positive and unlabeled instances.

In the final step, the predicted labels of all algorithms were compared with the ground truth labels to identify the instances that were identified as reliable negative instances by all three classifiers. Comparing these negative instances with their ground truth labels indicated that the proposed approach can reliably determine negative instances from unlabeled data, as no positive instances were predicted as negative by all three classifiers.

2.2.5 Anomaly Detection

Next, an LSTM network was trained using the known positive instances and reliable negative instances obtained. LSTM networks have been shown to provide good results in classifying sequential data [56]. As LSTM network works better with balanced data and should be trained with a sufficient number of positive and negative instances, so the data was resampled. Therefore, the reliable negative instances obtained from the first step were shuffled, and 30% were randomly selected to provide the same number as the positive instances. The LSTM network was trained with 90% of this data, and validated with the remaining 10%.

The LSTM network employs the Sigmoid activation function, binary cross entropy loss function and Adam optimizer as they provide good performance and fast convergence in deep learning algorithms [60]. For the BGL and Thunderbird datasets, three hidden layers of size 256, batch size 128 and a maximum of 10 training epochs were used. Moreover, dropout with probability 0.5 and early stopping was used to prevent overfitting.

For the Openstack dataset, a single-layer network with embedding dimension 512 was used to train the network due to the smaller size compared to the other two datasets. The rest of the architecture for the Openstack dataset is the same as above. Note that all network parameters were chosen based on the experimental results obtained during the execution of the model. The proposed model algorithm and architecture are given in Algorithm 1 and Figure 2.3.

Algorithm 1 Proposed Model

Require: Dataset D scaled between $[0, 1]$.

Randomly select a small portion of positive samples P and remove the labels of the remaining samples to obtain the unlabeled set U.

Split U into training and test sets with 95% test for Openstack, and 99% test for BGL and Thunderbird.

Use IF to classify the training set.

Use PU Learning with bagging to classify the training set.

Use RF to classify the training set.

Choose the samples all algorithms predicted as negative as the reliable negative samples.

Create a balanced data set from the known positive and reliable negative samples and use this to train the LSTM network.

Evaluate the trained LSTM network using the test data.

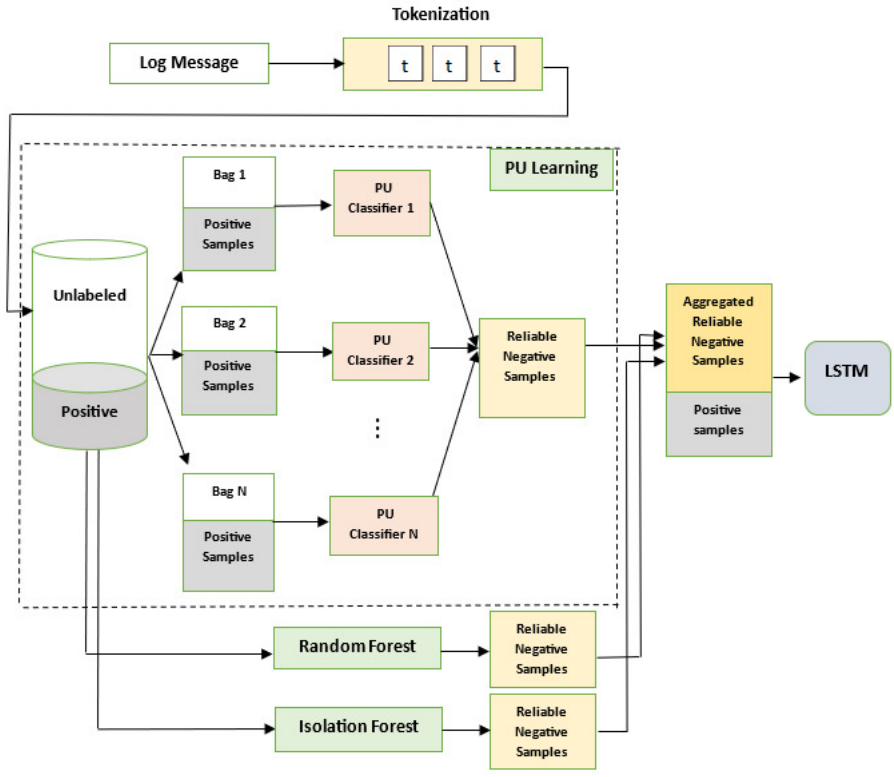

Figure 2.3: Model training pipeline including data preprocessing.

2.3 Results

In this section, the proposed model is evaluated using the performance metrics precision, recall, F-measure and accuracy. They are based on the confusion matrix shown in Figure 2.4. Precision measures the accuracy of the positive predictions, while recall measures the completeness of the positive predictions. Precision is expressed as

$$Precision = \frac{TP}{TP + FP}, \tag{2.1}$$

where TP is the number of positive logs correctly predicted and FP is the number of negative logs predicted to be positive. Recall (or sensitivity) is given by

$$Recall = \frac{TP}{TP + FN}, \tag{2.2}$$

where FN is the number of positive logs predicted to be negative. F-measure is expressed as

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{2.3}$$

Finally, accuracy is defined as the proportion of true positives and true negatives among the total number of instances examined and is given by

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.4}$$

Python and Scikit-learn[1] were used to implement the algorithms. The experiments were repeated 10 times on the BGL, Openstack and Thunderbird datasets and the average results in terms of precision, recall and F-measure are summarized in Table 2.5.

[1] `https://github.com/scikit-learn/scikit-learn`

		Actual Class	
		Positive	Negative
Predicted Class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Figure 2.4: Confusion matrix for binary classification.

2.3.1 Openstack

The Openstack dataset has 18,434 positive and 137,074 negative logs. After randomly selecting 0.03% of positive (abnormal) logs from the dataset, and unlabeling the remaining logs (flipping the remaining 1 labels to 0), 8489 logs were used for training and the remaining 147,019 logs were used for testing. The experiment was repeated 10 times and the minimum, average and maximum scores for the Openstack dataset are given in Table 2.5. According to these results, the average precision, recall and F-measure were 98.9%, 98.6% and 98.8% for positive logs, and 99.1%, 99.4% and 99.3% for negative logs, respectively.

The precision, recall and F-measure results for positive log messages were slightly better than the 94%, 99% and 97% with the Deeplog network [61]. A similar level of accuracy was obtained compared with the results in [31] and [58] for Autoencoder networks and the Auto-B/LSTM and Auto-GRU deep learning approaches, respectively. Moreover, the results obtained in [62] using deep learning methods were slightly higher for positive logs with an average of 99.9%, 99.7% and 99.8% for precision, recall and F-measure, respectively, and very close for negative instances.

2.3.2 BGL

The BGL dataset has 348,460 positive and 4,399,502 negative log messages. After randomly selecting 0.01% of positive logs as the positive log dataset and unlabeling the rest of the data, the dataset was shuffled and split into 22,174 logs for training and 4,725,788 logs for testing.

The average precision, recall and F-measure were 99.1%, 98.7% and 99.1% for positive logs, and 98.4%, 99.1% and 98.7% for negative logs, respectively. These are

better than the results with the SVM unsupervised learning approach in [64] which had precision, recall and F-measure of 83%, 99% and 91%, respectively. Further, the 82.5%, 94.7% and 88.2% for the nLSALog algorithm [65] show that PU Learning can be more effective than other anomaly detection approaches. The method in [17] has a high F-measure of 99.4% which is slightly better than the average F-measure of 98.9% in this research.

The F-measure results in [13] are slightly better than those obtained here. The results are very close for positive logs, and slightly better for negative logs compared with the approach in [62] with average precision, recall and F-measure of 95.6%, 97.8% and 96.7% for negative logs and 99.8%, 99.6% and 99.7% for positive logs, respectively. The results for unsupervised learning in [31] are also close with average scores of 96.8%, 98.7% and 98.1% for negative logs, and 99.8%, 99.7% and 99.8% for positive logs, respectively.

2.3.3 Thunderbird

The Thunderbird data has 3,000,000 positive and 3,248,239 negative log messages. After randomly selecting 0.05% positive logs from the dataset and unlabeling the remaining, 62332 messages (1% of the dataset), were used for training and the rest for testing. The average precision, recall and F-measure for 10 runs are 97%, 98.4% and 97.7% for positive logs, and 98.4%, 99.1% and 99.1% for negative logs, respectively.

The method in [17] has an F-measure of 99.9% which is better than the average F-measure of 98.9% in this research. Comparing the results with those for the IKNN algorithm in [63] shows that the precision, recall and F-measure for positive logs are better than the 96%, 96% and 96% scores obtained in this work. On the other hand, the approach in [62] has average scores for positive logs of 98.9%, 99.6% and 99.3% for precision, recall and F-measure, respectively, which are slightly better than those for the proposed model.

Comparing the results with those in [58] for the Auto-LSTM, Auto-BLSTM and Auto-GRU deep learning methods indicates a similar accuracy in detecting anomalies. The results in [31] for the IF algorithm and two autoencoders in an unsupervised model are very close to those here with average precision, recall and F-measure of 97.2%, 98.6% and 98.4% for negative logs, and 99.7%, 99.4% and 99.6% for positive logs, respectively.

2.4 Discussion

In this research, an ensemble-based PU Learning method was designed to obtain reliable negative logs for training an anomaly detection model using unlabeled data instead of training with fully labeled logs.

2.4.1 Isolation Forest Performance

The proposed model involves the integration of three ML algorithms, namely RF, PU Learning and IF, to predict negative log messages. The general idea of the IF algorithm is that anomalies can be more easily separated (isolated) from the dataset given that they possess characteristics that are less likely to occur. Thus it learns the characteristics of normal logs in order to distinguish the abnormal logs. A contamination rate of 0.1 was chosen. Moreover, the number of base estimators in the ensemble was tuned based on an acceptable accuracy with the least number of positive instances.

The IF algorithm has other characteristics that make it a good candidate for anomaly detection. It does not assume a distribution and is able to detect anomalies at a multi-dimensional level. More importantly, it is computationally efficient with linear time complexity and low memory requirements. Therefore, it scales well to large datasets.

Table 2.3 shows that the IF algorithm did not provide good results finding negative instances among unlabeled data as the precision, recall and F-measure for positive log messages in all datasets were poor. However, the corresponding results for negative logs were much better. The corresponding results obtained with all three algorithms for the BGL dataset show an increase in the average precision, recall and F-measure for positive logs from 59.9%, 62.6% and 61.2% to 98.4%, 99.1% and 98.7%, respectively. For negative logs, these results were improved from 79.4%, 78.2% and 78.8% to 99.1%, 98.7% and 99.1%, respectively.

Table 2.3: Average precision, recall, and F-measure for 10 runs with the IF, PU Learning and RF classifiers for different datasets.

Classifier	Dataset	Label	Precision	Recall	F-measure
Isolation Forest	**BGL**	0	79.4%	78.2%	78.8%
		1	59.9%	62.2%	61.2%
	Openstack	0	68.1%	69.9%	68.9%
		1	71.1%	70.5%	70.8%
	Thunderbird	0	68.9%	69.2%	69.1%
		1	69.9%	67.3%	68.6%
PU Learning	**BGL**	0	90.1%	89.7%	89.8%
		1	91.9%	90.3%	91.1%
	Openstack	0	89.5%	90.6%	84.1%
		1	91.1%	90.8%	90.9%
	Thunderbird	0	89.2%	90.7%	89.9%
		1	88.5%	90.6%	89.5%
Random Forest	**BGL**	0	67.9%	69.4%	68.6%
		1	65.2%	66.1%	65.6%
	Openstack	0	70.1%	72.3%	71.2%
		1	70.4%	73.5%	71.9%
	Thunderbird	0	67.9%	68.2%	68.1%
		1	69.4%	68.8%	69.1%

2.4.2 PU Learning Performance

In this research, a weighted approach to PU Learning was considered where the weights represent the likelihood, or conditional probability, that an unlabeled instance x belongs to the positive set. A standard learning algorithm was then used with the weighted unlabeled instances to learn a classifier $f(x) = p(y = 1|x)$ for log anomaly detection.

The results in Table 2.3 indicate that PU Learning performs better than the other two algorithms. The precision, recall and F-measure for positive and negative log messages in all datasets were higher than with the IF and RF algorithms. Figure 2.5 shows that for the BGL dataset, the average F-measure with PU Learning for negative instances is about 15% higher than with the IF and RF algorithms, and about 20% higher for positive instances. Comparing these results with those in Table 2.5 indicates that using the IF, PU Learning and RF algorithm together improves model performance. The average precision, recall and F-measure were improved from 76.9%, 80.2% and 76.4% to 98.6%, 99.1% and 99.1% for negative logs, and from 74.2%, 78.1% and 75.4% to 97.6%, 98.5% and 98.5% for positive logs, respectively. These results demonstrate the impact of employing all three algorithms to train the LSTM network.

Employing bagging with PU Learning improved the prediction results through building estimators on different instances and taking a majority vote for classification. Bagging helps reduce the prediction variance. This is particularly beneficial in PU Learning where the lack of labeled negative instances can make models prone to overfitting. Although bagging generally reduces overfitting, there could be a point beyond which adding more bags does not yield significant improvements and may lead to increase computational complexity. Thus, there is a tradeoff between computational complexity and model performance. The results obtained show that about 1000 bags provides a good balance between performance and complexity as adding more bags beyond 1000 did not significantly improve the performance.

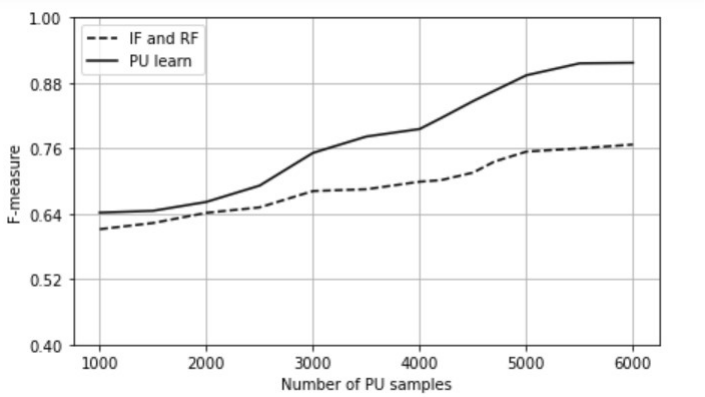

Figure 2.5: PU Learning performance with the IF and RF algorithms and BGL dataset for negative log messages.

2.4.3 LSTM Network Performance

In general, methods based on neural networks perform significantly better than most traditional methods [17]. The key difference between LSTMs and other types of neural networks is the way that they deal with information over time. LSTMs can process information recurrently, meaning that they can take input at one time step and use it to influence the output at future time steps. This allows LSTMs to learn from sequences of data.

There are a number of advantages that LSTMs have over traditional RNNs.

- LSTMs are much better at handling long-term dependencies. This is due to their ability to remember information for extended periods of time.
- LSTMs are less susceptible to the vanishing gradient problem. This is because they use a different kind of activation function, known as an LSTM cell, which helps to preserve information over long sequences.
- LSTMs are very efficient at modeling complex sequential data. This is because they can learn high-level representations that capture the structure of the data.

The number of LSTM layers depends on the complexity of the problem. More layers can capture more complex relationships but also increases the risk of overfitting and requires more computational power. In this research, an LSTM network with different numbers of layers was examined using the BGL, Openstack and Thunderbird datasets. Table 2.4 gives the average accuracy and running time for 3 runs of the

model. These results show that more than three layers produce more test errors compared to training errors and has a higher running time, which is not desirable.

The results from the BGL and Thunderbird datasets indicate that using three layers offers a good balance between accuracy and running time, while with the OpenStack dataset, a single layer is sufficient for satisfactory accuracy.

Table 2.4: Average F-measure, accuracy and training time for 3 runs of the LSTM network with various numbers of layers and 10 epochs.

Dataset	#Layers	Label	F-measure	Accuracy	Time (s)
BGL	1	0	80.4%	82.1%	1160
		1	79.9%	80.6%	
	2	0	88.5%	86.8%	1508
		1	89.1%	89.3%	
	3	0	98.9%	99.1%	2110
		1	97.9%	98.2%	
Openstack	1	0	97.1%	98.7%	98
		1	97.9%	98.1%	
	2	0	98.2%	98.9%	132
		1	98.3%	98.8%	
Thunderbird	1	0	67.9%	69.4%	1075
		1	65.2%	66.1%	
	2	0	86.3%	88.6%	1465
		1	87.3%	89.9%	
	3	0	97.9%	98.7%	2245
		1	99.1%	98.5%	

2.4.4 Model Performance

The proposed method has some advantages in comparison with other log message detection strategies. The first advantage is the ability of the model to deal with imbalanced data in identifying reliable negative instances. In log message datasets, the number of labeled logs is usually much smaller than unlabeled ones, as indicated in Table 2.1. As discussed in [67], imbalanced data have little impact on the accuracy of ensemble tree-based algorithms due to the fact that they rely on a majority vote of decision trees. The proposed method leverages tree-based algorithms to obtain negative instances among unlabeled data to create balanced training sets.

Another advantage of the proposed method is that only a small amount of training data was used (0.01% for the BGL, 0.03% for the Openstack and 0.05% for the Thunderbird datasets), whereas training models such as deep networks usually requires a larger amount of data for convergence. The Openstack dataset requires more training data because it is a small dataset (around 25 times smaller than Thunderbird and 30 times smaller than BGL).

Log preprocessing can impact the performance of the model. With text data, words can be converted to numbers using different methods such as Bag-of-Words, Term Frequency-Inverse Document Frequency (TF-IDF) or word frequency. In recent years, embedding methods such as Word2vec [1] and GloVe [2] have been proposed to deal with the relationship between words. In the proposed model, the original data was converted from text to digits using word frequency. This improved the training efficiency and shortened the training time. The proposed model had average training times of 103 s, 2107 s, and 2233 s after preprocessing for the Openstack, BGL, and Thunderbird datasets, respectively.

The three algorithms were evaluated separately. The results demonstrate that the misclassification rate was higher when they were individually employed to label data, while this rate was lower when all three classifiers were employed simultaneously. The results showed that although PU Learning performs better than the other two algorithms, using instances that all three algorithms agree on as reliable negative instances significantly improved the prediction results. Comparing the ground truth labels with the assigned labels indicated that no positive instances were wrongly predicted as negative.

The results obtained show that ensemble tree-based algorithms along with PU Learning can have a significant impact on the performance with large-scale unla-

beled datasets. Although these methods provide good results for classification tasks individually, they are better when used together.

Table 2.5: The precision, recall, F-measure, and average time (seconds), with the minimum, maximum and average (in brackets) values for the BGL, Openstack and Thunderbird datasets for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

Dataset	**Label**	**Precision**	**Recall**	**F-measure**	**Time (s)**
BGL	0	97.7%- 98.4% -99.1%	98.3%- 99.1% -99.8%	98.1% - 98.7% - 99.4%	2107
	1	97.9%-(**99.2%**)-99.9%	97.1%-(**98.7%**)-99.8%	98%-(**99.1%**)-99.9%	
Openstack	0	97.9%-99.1% -99.8%	98.3%-99.4% -99.9%	98.2%-99.3% -99.9%	103
	1	99%-(**98.9%**)-99.6%	98.2%-(**98.6%**)-99.1%	98.2%-(**98.8%**)-99.8%	
Thunderbird	0	97.9%-98.4%-99.1%	98.3%-99.1%-99.8%	98.1%-99.1%-99.4%	2233
	1	97.9%-(**97.1%**)-98.2%	97.1%-(**98.4%**)-99.8%	98.1%-(**97.7%**)-99.9%	

Chapter 3

Conclusion

In this chapter, the main results are summarized. This is followed by some potential areas for future work based on this research.

3.1 Conclusion

Log messages provide extensive event descriptions that can be very helpful for network and service management. However, due to the limited number of accessible labels, it is quite challenging to identify and classify log messages. In this thesis, a framework was proposed using three ML algorithms, namely Isolation Forest, PU Learning, and Random Forest. It was shown that PU Learning had a significant impact on training with a very limited number of labeled instances.

The proposed method exploits the strength of the Isolation Forest, PU Learning and Random Forest algorithms in distinguishing positive logs from other data in order to extract reliable negative instances. The results presented demonstrate that this approach is effective. An LSTM network was then trained using the positive and negative instances obtained.

The proposed model was evaluated using three well-known log datasets, namely Openstack, BGL, and Thunderbird. The results obtained show that this approach is at least as good as other well-known methods in terms of precision, recall and F-measure, and in some cases superior.

3.2 Future Work

A significant challenge in log message anomaly detection with PU Learning is the high-dimensional nature of log data. This data often contains a vast array of features, some of which may contribute little to model performance. Techniques such as dimensionality reduction or feature selection can be explored to improve the performance of the PU Learning model. Dimensionality reduction techniques have been used to distill vast datasets into more manageable forms without substantial loss of information, so they offer a promising avenue for improving the effectiveness and efficiency of ML models.

Feature selection methods are used to identify the most relevant features for learning tasks, and can play a crucial role in simplifying models and reducing computational complexity. Feature selection methods such as information gain [38], chi-squared test [45], and Recursive Feature Elimination (RFE) [41] have been used to identify the most relevant features. Adapting these methods for PU Learning in the context of high-dimensional log data is an interesting area for research. It can also be useful in preprocessing to understand the underlying structure of log data before applying PU Learning.

Bibliography

[1] Mikolov, T. and Chen, K. and Dean, J., "Efficient Estimation of Word Representations in Vector Space," arXiv preprint arXiv:1301.3781v3, 2013.

[2] Pennington, J. and Socher, R. and Manning, C. D., "Glove: Global Vectors for Word Representation," *Conference on Empirical Methods in Natural Language Processing*, pp. 1532-1543, 2014.

[3] Yuan, D. and Mai, H. and Xiong, W. and Tan, L. and Zhou, Y. and Pasupathy, S., "SherLog: Error Diagnosis by Connecting Clues from Run-time Logs," *Architectural Support for Programming Languages and Operating Systems*, pp. 143-154, 2010.

[4] Simeone, O., "A Very Brief Introduction to Machine Learning With Applications to Communication Systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648-664, 2018.

[5] Breiman, Leo., "Bagging Predictors," *Machine Learning*, vol. 24, pp. 123-140, 1996.

[6] Ward, G. and Hastie, T. and Barry, S. and Elith, J. and Leathwick, J. R., "Presence-only Data and the EM Algorithm," *Biometrics*, vol. 65, no. 2, pp. 554–563, 2009.

[7] Liu, Z. and Shi, D. and Qin, Q., "Partially Supervised Classification: Based on Weighted Unlabeled Samples Support Vector Machine," *International Journal of Data Warehousing and Mining*, vol. 1, no. 3, pp. 42–56, 2006.

[8] Jaskie, K. and Spanias, A.,"Positive and unlabeled learning algorithms and applications: A survey," *International Conference on Information, Intelligence, Systems and Applications*, 2019.

[9] Bekker, J. and Davis, J.,"Learning from Positive and Unlabeled Data: A Survey," *Machine Learning*, vol. 109, no. 4, pp. 719–760, 2020.

[10] Zhang, D. and Lee, W. S., "A Simple Probabilistic Approach to Learning from Positive and Unlabeled Examples," *UK Workshop on Computational Intelligence, Springer, Berlin, Germany*, pp. 83–87, 2005.

[11] Tax, D. MJ. and Duin, R. PW., "Uniform Object Generation for Optimizing One-class Classifiers," *Journal of Machine Learning Research*, vol. 2, pp. 155-173, 2001.

[12] Manevitz, L. M. and Yousef, M., "One-class SVMs for Document Classification," *Journal of Machine Learning Research*, vol. 2, pp. 139-154, 2001.

[13] Meng, W. and Liu, Y. and Zhang, S. and Zaiter, F. and Zhang, Y. and Huang, Y. and Yu, Z. and Zhang, Y. and Song, L. and Zhang, M. and Pei, D., "LogClass: Anomalous Log Identification and Classification with Partial Labels," *IEEE Transactions on Network and Service Management*, vol. 18, no. 21, pp. 1870-1884, 2021.

[14] Menon, A. K. and Van Rooyen, B. and Oong, C. S. and Williamson, R. C., "Learning from Corrupted Binary Labels via Class-Probability Estimation," *International Conference on Machine Learning, Proceedings of Machine Learning Research*, pp. 125-134, 2015.

[15] Lee, W. S. and Liu, B., "Learning with Positive and Unlabeled Examples using Weighted Logistic Regression," *International Conference on Machine Learning, Washington, DC, USA*, pp. 448–455, 2003.

[16] Sansone, E. and De Natale, F. G. B. and Zhou, Z. H., "Efficient Training for Positive Unlabeled Learning," *IEEE Transactions on Pattern Analysis and Machine*, vol. 41, no. 11, pp. 2584-2598, 2019.

[17] Wittkopp, T. and Scheinert, D. and Wiesner, P. and Acker, A. and Kao, O., "PULL: Reactive Log Anomaly Detection based on Iterative PU Learning," arXiv preprint arXiv:2301.10681, 2023.

[18] Hou, M. and Chaib-Draa, B. and Li, C. and Zhao, Q., "Generative Adversarial Positive-Unlabelled Learning," arXiv preprint arXiv:1711.08054, 2018.

[19] Bekker, J. and Davis, J., "Learning from Positive and Unlabeled Data: A Survey," arXiv preprint arXiv:1811.04820, 2018.

[20] He, F. and Liu, T. and Webb, G. I. and Tao, D., "Instance-dependent PU Learning by Bayesian Optimal Relabeling," arXiv preprint arXiv:1808.02180, 2018.

[21] Yang, P. and Li, X. and Chua, H. and Kwoh, C. K. and Ng, S. K., "Ensemble Positive Unlabeled Learning for Disease Gene Identification," *Bioinformatics*, vol. 28, no. 20, pp. 2640–2647, 2014.

[22] Liu, F. T. and Ting, K. M. and Zhou, Z. H., "Isolation Forest," *IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[23] Li, X. L. and Liu, B., "Learning from Positive and Unlabeled Examples with Different Data Distributions," *Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Springer, Berlin, Germany*, pp. 218–229, 2005.

[24] Hariri, S. and Carrasco Kind, M. and Brunner, R. J., "Extended Isolation Forest," arXiv preprint arXiv:1811.02141, 2018.

[25] Reidemeister, T. and Jiang, M. and Ward, P. A. S., "Mining Unstructured Log Files for Recurrent Fault Diagnosis," *International Symposium on Integrated Network Management and Workshops*, pp. 377–384, 2011.

[26] Wang, J. and Tang, Y. and He, S. and Zhao, C. and Sharma, P.K. and Alfarraj, O. and Tolba, A., "LogEvent2vec: LogEvent-to-vector based Anomaly Detection for Large-scale Logs in Internet of Things," *Sensors*, vol. 20, no. 9, art. 2451, 2020.

[27] Wang, M. and Xu, L. and Guo, L., "Anomaly Detection of System Logs based on Natural Language Processing and Deep Learning," *International Conference on Frontiers of Signal Processing*, pp. 140-144, 2018.

[28] Tao, X. and Peng, . and Zhao, F. and Zhao, P. and Wang, Y., "A Parallel Algorithm for Network Traffic Anomaly Detection based on Isolation Forest," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, pp. 1-11, 2018.

[29] Nissim, N. and Moskovitch, R. and Rokach, L. and Elovici, Y., "Detecting Unknown Computer Worm Activity via Support Vector Machines and Active Learning," *Pattern Analysis and Applications*, vol. 15, no. 4, pp. 459–475, 2012.

[30] Taylor, A. and Japkowicz, N. and Leblanc, S., "Frequency-Based Anomaly Detection for the Automotive CAN Bus," *World Congress on Industrial Control Systems Security*, pp. 45-49, 2015.

[31] Farzad, A. and Gulliver, T. A., "Unsupervised Log Message Anomaly Detection," *Information and Communications Technology Express*, vol. 6, no. 3, pp. 229-237, 2020.

[32] Li, X. and Liu, B., "Learning to Classify Texts using Positive and Unlabeled Data," *International Joint Conference on Artificial Intelligence*, pp. 587–592, 2003.

[33] Denis, F. and Gilleron, R. and Letouzey, F., "Learning from Positive and Unlabeled Examples," *Theoretical Computer Science*, vol. 348, no. 1, pp. 70-83, 2005.

[34] De Comité, F. and Denis, F. and Gilleron, R. and Letouzey, F., "Positive and unlabeled examples help learning," *International Conference of Algorithmic Learning Theory, Tokyo, Japan*, pp. 219-230, 1999.

[35] Joachims, T., "A Probabilistic Analysis of the Rocchio Algorithm With TFIDF For Text Categorization," *International Conference on Machine Learninge, Nashville, TN, USA*, pp. 143–151, 1997.

[36] Schölkopf, B. and Platt, J. C. and Shawe-Taylor, J. and Smola, A. J., "Estimating the Support of a High-Dimensional Distribution," *Neural Computing*, pp. 1443-1471, 2001.

[37] Liu, B. and Dai, Y. and Li, X. and Lee, W. S. and Yu, P. S., "Building Text Classifiers using Positive and Unlabeled Examples," *IEEE International Conference on Data Mining*, pp. 179-186, 2003.

[38] Azhagusundari, B. and Thanamani, A. S., "Feature Selection based on Information Gain," *International Journal of Innovative Technology and Exploring Engineering*, vol. 2, no. 2, pp. 18-21, 2013.

[39] Yu, H. and Han, J. and Chang, K. C. C., "PEBL: Web Page Classification without Negative Examples," *IEEE Transaction on Knowledge and Data Engineering*, vol. 16, no. 1, pp. 70–81, 2004.

[40] Elkan, C. and Noto, K., "Learning Classifiers from Only Positive and Unlabeled Data," *International Conference on Knowledge Discovery and Data Mining, New York, NY, USA*, pp. 213–220, 2008.

[41] Darst, B. F. and Malecki, K. C. and Engelman, C. D., "Using Recursive Feature Elimination in Random Forest to Account for Correlated Variables in High Dimensional Data," *BMC Genetics*, vol. 19, art. 65, 2018.

[42] Lee, W.S. and Liu, B., "Learning With Positive and Unlabeled Examples using Weighted Logistic Regression," *International Conference on Machine Learning, AAAI Press*, pp. 448–455, 2003.

[43] He, P. and Zhu, J. and He, S. and Li, J. and Lyu, MR., "An Evaluation Study on Log Parsing and Its Use in Log Mining," *IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 654–661, 2016.

[44] Lasisi, A. and Attoh-Okine, N., "Machine Learning Ensembles and Rail Defects Prediction: Multilayer Stacking Methodology," *Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, vol. 5, no. 4, art. 04019016, 2019.

[45] McHugh, M. L., "The Chi-square Test of Independence," *Biochemia Medica*, vol. 23, no. 2, pp. 143-149, 2013.

[46] Galar, M. and Fernandez, A. and Barrenechea, E. and Bustince, H. and Herrera, F., "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-based Approaches," *IEEE Transaction on Systems, Man, and Cybernetic*, vol. 42, no. 4, pp. 463-484, 2012.

[47] Fawcett, T., "An Introduction to ROC Analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, 2006.

[48] Geurts, P. and Ernst, D. and Wehenkel, L., "Extremely Randomized Trees," *Machine Learning*, vol. 63, no. 1, pp. 3-42, 2006.

[49] Liu, F. T. and Ting, K. M. and Zhou, Z. H., "Isolation-based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, pp. 31-39, 2012.

[50] Suykens, J.A.K. and Vandewalle, J., "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters*, vol. 9, pp. 293-300, 1999.

[51] Yu, H., "Single-class Classification With Mapping Convergence," *Machine Learning*, pp. 49-69, 2005.

[52] Mordelet, F. and Vert, J. P., "A Bagging SVM to Learn from Positive and Unlabeled Examples," *Pattern Recognition Letters*, vol. 37, pp. 201-209, 2012.

[53] Zhao, Y. and Xu, Q. and Jiang, Y. and Wen, P. and Huang, Q., "Dist-PU: Positive-Unlabeled Learning from a Label Distribution Perspective," *Computer Vision and Pattern Recognition*, pp. 14441-14450, 2022.

[54] Cutler, A. and Cutler, D. R. and Stevens, J. R., "Random Forests," *Ensemble Machine Learning: Methods and Applications*, pp. 157-175, 2012.

[55] Hochreiter, S. and Schmidhuber, J., "Long Short-term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 2009.

[56] Graves, A., "Supervised Sequence Labeling with Recurrent Neural Networks," *Studies in Computational Intelligence, Springer, Berlin, Germany*, pp. 769–789, 2012.

[57] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., Williamson, R. C. "Estimating the Support of a High-dimensional Distribution," *Neural Computation*, vol. 13. no. 7, pp. 1443-1471, 2001.

[58] Farzad, A. and Gulliver, T. A., "Log Message Anomaly Detection and Classification using Auto-B/LSTM and Auto-GRU," arXiv preprint arXiv:1911.08744, 2019.

[59] Du, W. and Zhu, Z. and Wang, C. and Yue, Z., "The Real-time Big Data Processing Method Based on LSTM for the Intelligent Workshop Production Process," *IEEE International Conference on Big Data Analytics*, pp. 63-67, 2020.

[60] Sun, Y. and Xu, W. and Zhang, J. and Xiong, J. and Gui, G., "Super-resolution Imaging using Convolutional Neural Networks," *Communications, Signal Processing, and Systems, Springer Nature, Berlin, Germany*, pp. 59–66, 2020.

[61] Du, M. and Li, F. and Zheng, G. and Srikumar, V., "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," *ACM Conference on Computer and Communications Security*, pp. 1285-1298, 2017.

[62] Farzad, A. and Gulliver, T. A., "Two Class Pruned Log Message Anomaly Detection," *SN Computer Science*, vol. 2, no. 5, pp. 391, 2021.

[63] Wang, B. and Ying, S. and Cheng, G. and Wang, R. and Yang, Z. and Dong, B., "Log-Based Anomaly Detection With the Improved K-Nearest Neighbor," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 2, pp. 239-262, 2020.

[64] He, S. and Zhu, J. and He, P. and Lyu, M. R., "Experience Report: System Log Analysis for Anomaly Detection," *IEEE International Symposium on Software Reliability Engineering*, pp. 207-218, 2016.

[65] Yang, R. and Qu, D. and Gao, Y. and Qian, Y. and Tang, Y., "nLSALog: An Anomaly Detection Framework for Log Sequence in Security Management," *IEEE Access*, vol. 7, pp. 181152-181164, 2019.

[66] Farzad, A. and Gulliver, T. A., "Log Message Anomaly Detection and Classification using Auto-B/LSTM and Auto-GRU," arXiv preprint arXiv:1911.08744, 2019.

[67] Liu, D. and Zhao, Y. and Xu, H. and Sun, Y. and Pei, D. and Luo, J. and Jing, X. and Feng, M., "Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning," *Internet Measurement Conference*, pp. 211-224, 2015.