Log Message Anomaly Detection Using Machine Learning

by

Amir Farzad
M.Sc., Shahrood University of Technology, 2016

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

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Log Message Anomaly Detection Using Machine Learning

by

Amir Farzad
M.Sc., Shahrood University of Technology, 2016

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)

Dr. Miguel Nacenta, Outside Member
(Department of Computer Science)

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)

Dr. Miguel Nacenta, Outside Member
(Department of Computer Science)

ABSTRACT

Log messages are one of the most valuable sources of information in the cloud and other software systems. These logs can be used for audits and ensuring system security. Many millions of log messages are produced each day which makes anomaly detection challenging. Automating the detection of anomalies can save time and money as well as improve detection performance. In this dissertation, Deep Learning (DL) methods called Auto-LSTM, Auto-BLSTM and Auto-GRU are developed for log message anomaly detection. They are evaluated using four data sets, namely BGL, Openstack, Thunderbird and IMDB. The first three are popular log data sets while the fourth is a movie review data set which is used for sentiment classification. The results obtained show that Auto-LSTM, Auto-BLSTM and Auto-GRU perform better than other well-known algorithms.

Dealing with imbalanced data is one of the main challenges in Machine Learning (ML)/DL algorithms for classification. This issue is more important with log message data as it is typically very imbalanced and negative logs are rare. Hence, a model is proposed to generate text log messages using a Sequence Generative Adversarial Network (SeqGAN) network. Then features are extracted using an Autoencoder and anomaly detection is done using a GRU network. The proposed model is evaluated with two imbalanced log data sets, namely BGL and Openstack. Results are presented which show that oversampling and balancing data increases the accuracy of anomaly detection and classification.

Another challenge in anomaly detection is dealing with unlabeled data. Labeling even a small portion of logs for model training may not be possible due to the

high volume of generated logs. To deal with this unlabeled data, an unsupervised model for log message anomaly detection is proposed which employs Isolation Forest and two deep Autoencoder networks. The Autoencoder networks are used for training and feature extraction, and then for anomaly detection, while Isolation Forest is used for positive sample prediction. The proposed model is evaluated using the BGL, Openstack and Thunderbird log message data sets. The results obtained show that the number of negative samples predicted to be positive is low, especially with Isolation Forest and one Autoencoder. Further, the results are better than with other well-known models.

A hybrid log message anomaly detection technique is proposed which uses pruning of positive and negative logs. Reliable positive log messages are first identified using a Gaussian Mixture Model (GMM) algorithm. Then reliable negative logs are selected using the K-means, GMM and Dirichlet Process Gaussian Mixture Model (BGM) methods iteratively. It is shown that the precision for positive and negative logs with pruning is high. Anomaly detection is done using a Long Short-Term Memory (LSTM) network. The proposed model is evaluated using the BGL, Openstack, and Thunderbird data sets. The results obtained indicate that the proposed model performs better than several well-known algorithms.

Last, an anomaly detection method is proposed using radius-based Fuzzy C-means (FCM) with more clusters than the number of data classes and a Multilayer Perceptron (MLP) network. The cluster centers and a radius are used to select reliable positive and negative log messages. Moreover, class probabilities are used with an expert to correct the network output for suspect logs. The proposed model is evaluated with three well-known data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model provides better results than existing methods.

Table of Contents

List of Acronyms

ANN	Artificial Neural Network
BC	Bray-Curtis
BGL	BlueGene/L
BGM	Dirichlet Process Gaussian Mixture Model
BLSTM	Bidirectional Long-Short Term Memory
CAN	Controller Area Network
CNN	Convolutional Neural Network
DL	Deep Learning
DP	Dirichlet Process
E-step	Expectation step
EEnvelope	Elliptical Envelope
EM	Expectation Maximization
FCM	Fuzzy C-means
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
IKNN	Improved K-Nearest Neighbors
kd-tree	K-dimensional tree search

KNN	K-Nearest Neighbors
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
M-step	Maximization step
MC	Monte Carlo
ML	Machine Learning
MLP	Multilayer Perceptron
NLP	Natural Language Processing
OC-SVM	One-Class Support Vector Machine
PCA	Principal Component Analysis
PDF	Probability Density Function
PUL	Positive and Unlabeled Learning
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SeqGAN	Sequence Generative Adversarial Network
SOM	Self-Organizing Map
SVM	Support Vector Machine
Tanh	Tangent hyperbolic

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude and appreciation to Dr. Aaron Gulliver;
A great supervisor and human being. Thank you for all the help from the beginning.
Further, thanks to my committee members Dr. Li and Dr. Nacenta.
I want to thank Janice Closson, Ashleigh Carlsen and Dan Mai who always helped me.
And I would like to thank my mom, dad, sister and grandma (who I miss a lot) for all the support.

Dedication

To my mom and dad.
And to a friend.

Everything in the Universe is within you.
Ask all from yourself.
-"Rumi"

Chapter 1

Introduction

Most organizations now use cloud or application servers. Companies and consumers demand 24/7 accessibility to their cloud and application services as connection failures can severely affect operations. Logging is a method of maintaining information for audits and ensuring system security [110]. Log messages that show the state of the system are employed to improve efficiency and reliability. They play an important role in cloud systems such as Google Cloud Platform, Amazon Web Services and Microsoft Azure because they are used for system maintenance [29]. Runtime information captured by log message is used by developers to track and debug software systems [110]. An example of a log message is shown in Fig 1.1. Log messages are unstructured text consisting of time stamps, verbosity level and raw content regarding the system status. The composition of log messages can vary considerably which makes it difficult to identify anomalies [103].

Log messages are used for a variety of purposes such as event and anomaly detection [95], [88]. Most strategies use rules to detect anomalies but this requires a priori knowledge [99]. Some consider only one characteristic, such as time stamps, which restricts the ability to detect anomalies. Anomaly detection can be done manually, but this is not feasible for large systems due to the volume of data produced and the

Log message	2015-10-18 19:16:12 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-2680500627064966252 terminating
Time stamp	2015-10-18 19:16:12
Verbosity level	INFO
Raw content	dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-2680500627064966252 terminating

Figure 1.1: An example of a log message which consists of time stamp, verbosity level and raw content.

complexity of the log messages [48]. As a consequence, automated approaches to log message analysis are needed to accurately detect anomalies in a timely manner.

The ability of a program to learn and develop algorithmic solutions for tasks can be described as Machine Learning (ML) [78]. ML relies on past tests, while statistical methods typically rely on understanding the rules that produced the data. ML algorithms change their strategy on the basis of new information received. ML approaches fall into three main categories: supervised, unsupervised and Reinforcement Learning (RL) [26]. In supervised learning, a labeled data set is needed. This data set must be obtained manually and split into two sets (training and testing), or three sets (training, validation and testing). The training set is used to train the algorithm, the validation set is used to validate and tune the algorithm, and the testing set is used to test unseen data for prediction. Decision tree and naïve Bayes are two well-known supervised ML algorithms [57]. Unsupervised learning is used when the data set is unlabeled. This method learns from a small number of features and the trained model is used to recognize new data. Reinforcement Learning uses an agent and environment with state, action and reward to make decisions. Monte Carlo (MC) is an important RL algorithm which has been used for tasks such as playing games [17].

ML algorithms can be used in various tasks. An Elman recurrent network has been used for the classification of sequences, but it is computationally complex and hard to train [19], [15]. Another approach is a Self-Organizing Map (SOM) based on competition [41]. Positive and Unlabeled Learning (PUL) is an ML technique

that can be used for one-class classification where just positive data is available. PUL provides good results for tasks such as learning word embedding [39] and text classification [102].

Deep Learning (DL) is an ML technique that employs networks with multiple connected layers. DL can reduce data complexity and find similarities between data [21]. Thus, DL techniques are very suitable for big data. They can also be used for feature extraction and to reduce data dimensionality [38]. DL has produced excellent results for applications in image processing, text classification and Natural Language Processing (NLP) [100]. These techniques can be classified as generative, discriminative or hybrid. Discriminator methods are typically used for supervised classification. Generative methods are unsupervised and used to reveal patterns in unlabeled data. Hybrid methods combine discriminative and generative methods [36].

A Multilayer Perceptron (MLP) is a feedforward Neural Network (NN) which uses backpropagation for training [75]. MLP networks have been widely used in tasks such as emotion recognition [69] and image classification [106]. An Autoencoder [75] is a feedforward Artificial Neural Network (ANN) that can learn features from data and the data structure [90], [73]. Autoencoders have been applied to many different tasks such as probabilistic and generative modeling [40], [72], representation learning [31], and interpolation [62], [74].

A Long Short-Term Memory (LSTM) network [32] is a Recurrent Neural Network (RNN) that uses a cell to retain sequence information and remember long-term dependencies. LSTMs have been successfully used for tasks such as language modeling [104], translation [54], analysis of audio and video data [60], [13], phoneme classification [24], online mode-detection [68], emotion recognition [94] and acoustic modeling [76]. An LSTM network only utilizes past context but a Bidirectional Long-Short Term Memory (BLSTM) network [25] can utilize both past and future

contexts by processing the input data in both the forward and backward directions. BLSTM networks have been used for numerous tasks such as text recognition and classification [8], [109]. A Gated Recurrent Unit (GRU) [10] is similar to an LSTM network and the performance is comparable. However, it has been shown to provide better results than an LSTM in tasks such as speech recognition [37].

ML and DL algorithms have been used to develop a range of anomaly detection methods. Elliptical Envelope (EEnvelope) creates an elliptical area around the data mass center which is used in redshift estimation to detect anomalies [35] and to detect anomalies in audio sensors [4]. Local Outlier Factor (LOF) uses local data deviation to detect network flow anomalies to reduce the risk of internet attacks [70] and to detect anomalies in data traffic [56]. Support Vector Machine (SVM) and One-Class Support Vector Machine (OC-SVM) have been employed to detect unknown computer activity [66], anomalies in networks [63], [108] and automotive Controller Area Network (CAN) bus control networks [86].

A decision tree model was considered in [71] to detect faults using log messages. An improved supervised K-Nearest Neighbors (KNN) method was employed in [91] to detect anomalies in log messages. A hybrid model was presented in [64] to detect anomalies using K-means clustering and decision trees. The anomaly detection method proposed in [58] employs stacked Long Short-Term Memory (LSTM) networks. An online method which employs an LSTM network was given in [87] which provides better results than Isolation Forest. Deeplog [14] uses an LSTM network for anomaly detection but it has high computational complexity [84]. First, each log is mapped to its print statement (in the source code), using a log parser. Thus, each log is represented by a number and a session of logs is parsed to a sequence of numbers. Then an LSTM network is trained as a multi-class classifier using the window method [44] with normal sequences (corresponding to normal system operation). The

trained LSTM network is used to predict the probabilities of numbers occurring at a given time step. If the actual number is unlikely to occur based on the LSTM prediction, then it is considered to be an anomaly.

Generally, log messages are imbalanced because most indicate that the system is working properly and only a small portion indicates a significant problem. Data with a very unequal number of samples for the labels is called imbalanced. The problem of imbalanced data has been considered in tasks such as text mining [65], face recognition [53] and software defect prediction [77].

The imbalanced nature of log messages is one of the challenges for classification using DL. In binary classification, there are only two labels, and with imbalanced data most are normal (denoted major) logs. The small number of abnormal (denoted minor) logs makes classification difficult and can lead to poor accuracy with DL algorithms. This is because the normal logs dominate the abnormal logs. Oversampling and undersampling are two methods which can be used to address this problem. In undersampling, the major label samples are reduced so the number is similar to the number of minor label samples. A drawback of undersampling is loss of information [9]. In oversampling, the number of minor label samples is increased so it is similar to the number of major label samples. Recently, a generative adversarial network (GAN) [22] was proposed for generating images and showed good results in producing data which is similar to actual data [46]. GANs are able to generate more abstract and varied data than other algorithms [52].

Another challenge in anomaly detection is dealing with unlabeled data. Millions of log messages are produced each day in cloud and other systems, so labeling even a small percentage of these logs for model training is not possible. One way to deal with unlabeled data is to use unsupervised methods. Isolation Forest [50] is an ensemble approach that isolates abnormal samples to detect anomalies. It has been used

for tasks such as anomaly detection in sensor data [33] and intrusion detection [61]. Isolation Forest has high accuracy and linear time complexity, but it may perform poorly with large and complex data [85]. Recently, Extended Isolation Forest was proposed in [27] to improve anomaly detection with Isolation Forest. Extended Isolation Forest employs slopes and intercepts whereas Isolation Forest uses attributes and values. Functional Isolation Forest was proposed in [80] to detect anomalies in functional data using Isolated Forest. In this model, anomalies are detected using a collection of Functional Isolation Trees [80].

K-means is a well-known unsupervised clustering method which has been widely used in tasks such as detecting network intrusions [83], disease recognition [2] and anomaly detection in wireless sensor networks [92]. Fuzzy C-means (FCM) is another clustering algorithm. Unlike hard clustering methods such as K-means, FCM allows data to belong to more than one cluster. It has been used in tasks such as image segmentation [107] and collaborative filtering [42]. FCM uses membership (fuzzy membership) to determine the cluster class for a sample. Gaussian Mixture Model (GMM) is an unsupervised clustering method that assumes the data comes from a combination of Gaussian distributions. It has been used to solve problems such as detecting anomalies in flight operation data [47], intrusion detection [5] and real-time anomaly detection [12]. The Dirichlet Process Gaussian Mixture Model with variational inference (BGM) is a Bayesian mixture model (an extension of finite mixture models), which has been used for tasks such as anomaly detection in hyperspectral data [89].

1.1 Motivation and Problem Statement

Based on the previous literature review there is a gap for log message anomaly detection using different ML/DL methods, such as supervised, unsupervised and hybrid. In this dissertation, several methods which are suitable for log message anomaly detection are proposed. Although, most log message anomaly detection methods use log parsers, an NLP method for logs is suitable because logs are mostly text. Additionally, feature extraction methods for log messages can be used to improve anomaly detection. Most log message anomaly detection methods are supervised, however, this approach is not practical. This is because labeling is a time-consuming and costly task and not always possible due to the high volume of logs. Thus, unsupervised methods can be used to overcome the labeling problem. Moreover, hybrid methods can be used to select reliable logs and detect anomalies. In addition to accuracy, the speed of the algorithm is very important. This is because training in ML/DL methods can be time-consuming so complex algorithms for anomaly detection may not be practical. Thus, fast algorithms for log message anomaly detection are desirable.

The following problems and questions exist in log message anomaly detection.

1. Lack of proper pre-processing for text log messages for ML/DL algorithms.

2. How to develop supervised log message anomaly detection using feature extraction methods?

3. What is the effect of balancing logs for supervised log message anomaly detection?

4. How to develop an unsupervised method with high accuracy for log message anomaly detection?

5. How to select positive and negative logs unsupervised and perform anomaly detection using a hybrid method?

6. How can a fast algorithm using a very small amount of labeled data be developed for log message anomaly detection?

7. How can an expert be used to increase the performance of log message anomaly detection?

The goal in Chapter 2 is to develop a supervised model for log message anomaly detection. Problems 1 and 2 are considered in this chapter. The features are extracted using an Autoencoder and anomalies are detected using RNN-based models. In Chapter 3, a model is presented for oversampling log messages with RL to deal with imbalanced data and improve the supervised model results. Problem 3 is addressed in this chapter. The effect of balancing log messages for anomaly detection is investigated. To overcome the problem of unlabeled data, a model is proposed in Chapter 4 to detect anomalies with unsupervised learning. Problem 4 is considered in this chapter. The anomalies are detected using two Autoencoders and Isolation Forest. In Chapter 5, log messages are selected using a hybrid model to detect anomalies. Problem 5 is addressed in this chapter. The reliable logs are selected using unsupervised models. An LSTM network is used with selected reliable logs for anomaly detection. In Chapter 6, the goal is to detect anomalies in log message data using only a very small amount of labeled data for training to achieve a fast algorithm. Problems 6 and 7 are considered in this chapter. An expert is used to correct suspect outputs to improve the results.

1.2 Contributions

1. A text pre-processing method is presented for log messages instead of using log parsing methods.

2. The Auto-LSTM, Auto-BLSTM and Auto-GRU models are proposed for feature extraction and anomaly detection. The last layer output is used to extract features in Autoencoders.

3. A model is proposed for log message oversampling for anomaly detection using SeqGAN, Autoencoder, and GRU networks. Model results with and without log message oversampling are compared.

4. Isolation Forest is used to find positive logs rather than anomalies. An Autoencoder network is used to extract features for Isolation Forest which improves Isolation Forest results. The number of negative logs predicted to be positive (F_p) is low with Isolation Forest, particularly with one Autoencoder. Thus, most logs which are predicted to be positive are correct.

5. An unsupervised algorithm is presented which uses a GMM method to select reliable positive logs. An unsupervised algorithm is presented which employs K-means, GMM, and BGM methods iteratively to select reliable negative logs. Moreover, an LSTM network is used with the pruned logs for anomaly detection.

6. FCM with more clusters than the number of data classes is used with the training data to determine the cluster centers. The cluster centers and a radius are used to select reliable positive and negative logs with a kd-tree. Further, an MLP network is used for log anomaly detection using reliable logs.

7. Class probabilities are used with an expert to correct the network outputs for suspect logs.

1.3 Organization

The organization of this dissertation is as follows.

In Chapter 2, Auto-LSTM, Auto-BLSTM and Auto-GRU models are proposed for anomaly detection and classification. The proposed models are evaluated using four data sets and the results are compared with those for other well-known models. Section 2.1 presents the Autoencoder, LSTM, BLSTM and GRU architectures followed by the proposed models. Simulation results and a discussion are given in Section 2.2.

In Chapter 3, a model is proposed for log message oversampling for anomaly detection. The proposed model is evaluated using two well-known data sets and the results with and without oversampling are compared. In Section 3.1, the SeqGAN architecture is presented and the proposed model is described. Experimental results and a discussion are given in Section 3.2.

In Chapter 4, a model is proposed for unsupervised anomaly detection using Isolation Forest and two deep Autoencoder networks. The proposed model is evaluated using three log message data sets. In Section 4.1, the Isolation Forest architecture is presented and the proposed model is described. Experimental results and a discussion are given in Section 4.2.

In Chapter 5, a hybrid model using DL is proposed with pruning of positive and negative log messages. An unsupervised algorithm with a GMM is used to prune positive logs, and an unsupervised algorithm is used to prune negative logs using the K-means, GMM, and BGM methods iteratively. The proposed model is evaluated using three log message data sets. In Section 5.1, the K-means, GMM and BGM

architectures are presented and the proposed model is described. Experimental results and a discussion are given in Section 5.2.

In Chapter 6, radius-based FCM with more clusters than the number of data classes and an MLP network are employed for anomaly detection. Class probabilities with an expert are introduced to correct suspect outputs. The proposed model is evaluated with three log data sets. In Section 6.1, the FCM and MLP architectures are presented and the proposed model is described. Experimental results and a discussion are given in Section 6.2.

Finally, some concluding remarks and suggestions for future work are given in Chapter 7.

Chapter 2

Log Message Anomaly Detection and Classification Using Auto-B/LSTM and Auto-GRU

Log messages can be considered as sequences so an LSTM or GRU network with an Autoencoder is a suitable structure for anomaly detection. Thus in this chapter, models are proposed called Auto-LSTM, Auto-BLSTM and Auto-GRU that first extract features from log messages using an Autoencoder and then use the resulting extracted features in an LSTM, BLSTM or GRU network for anomaly detection and classification. In an Autoencoder network, the code layer (a hidden layer) is used for feature extraction and dimensionality reduction so the size of this layer is typically small [3]. However, in this chapter the last layer output is used to extract features so the code layer need not be smaller than the input. In addition, most state-of-the-art ML/DL methods for log message anomaly detection such as Deeplog use parsing models, but in the proposed model text logs with only simple pre-processing are employed. The models are evaluated based on the accuracy, precision, recall and F-measure using

three labeled log message data sets, namely BlueGene/L (BGL)*, Openstack† and Thunderbird‡. The IMDB movie review data set§ is also considered for sentiment classification. It is shown that good results are obtained with the proposed models for all four data sets with the same configuration.

The rest of the chapter is organized as follows. Section 2.1 presents the Autoencoder, LSTM, BLSTM and GRU architectures followed by the proposed models. The simulation results and discussion are given in Section 2.2. Finally, Section 2.3 provides some concluding remarks.

2.1 System Model

In this section, the Autoencoder, LSTM, BLSTM and GRU architectures employed in the proposed models are described.

2.1.1 Autoencoder Architecture

An Autoencoder [75] is a feedforward multilayer neural network in which the number of input and output units is the same. Training is performed using a loss function to assure that the output is close to the input. The aim is to learn a compact representation while minimizing the error for the input data. A deep Autoencoder is an Autoencoder which has more than one hidden layer [45]. It can represent complex distributions if multiple encoder and decoder layers are employed. The encoder and decoder outputs are

$$y = a(Wx + b), \tag{2.1}$$

*https://github.com/logpai/loghub/tree/master/BGL
†https://github.com/logpai/loghub/tree/master/OpenStack
‡https://github.com/logpai/loghub/tree/master/Thunderbird
§https://ai.stanford.edu/~amaas/data/sentiment

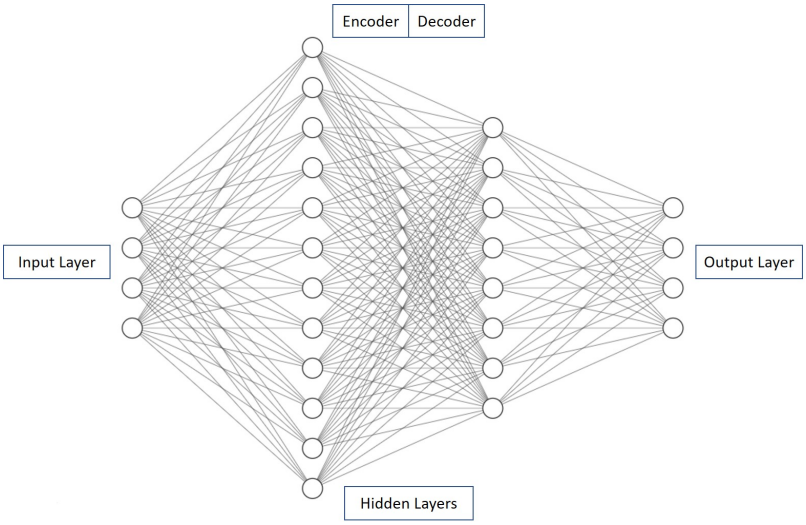

Figure 2.1: Autoencoder structure with an input layer, output layer and two hidden layers.

and

$$z = a(W'y + b'), \tag{2.2}$$

respectively, where x is the input, W is encoder weight matrix, b is encoder bias vector, W' is the decoder weight matrix, b' is decoder bias vector, and a is an activation function. Fig. 2.1 shows the architecture of an Autoencoder with an input layer, output layer, and two hidden layers.

2.1.2 LSTM Architecture

An LSTM is a recurrent neural network (RNN) [32] which has been successfully used to solve sequential data problems [23]. It has cells to store information in blocks which can be recurrently connected. These cells solve the vanishing gradient problem. Each LSTM block contains self-connected cells with input, forget, and output gates. These gates are designed to store information longer than feedforward neural networks to improve performance [23]. A block of an LSTM network contains recurrently connected cells as shown in Fig. 2.2. The cell input at time t is x_t and

the corresponding values for the input, forget and output gates are

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{2.3}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{2.4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{2.5}$$

respectively, where W and U are the weight matrices, b is the bias vector, and σ is the sigmoid activation function. The block input at time t is

$$\hat{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C), \tag{2.6}$$

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

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t, \tag{2.7}$$

where $\odot$ denotes point-wise multiplication. Finally, the block output at time t is given by

$$h_t = o_t \odot \tanh(C_t). \tag{2.8}$$

2.1.3 BLSTM Architecture

Bidirectional Long-Short Term Memory (BLSTM) consists of two LSTM networks. The input data is fed into two LSTM networks forwards and backwards with respect to time t, respectively, and both are connected to the same output layer. BLSTMs have the benefit of using both the past context and future context in a sequence. In a BLSTM, the forward and backward LSTM outputs at time t are concatenated which

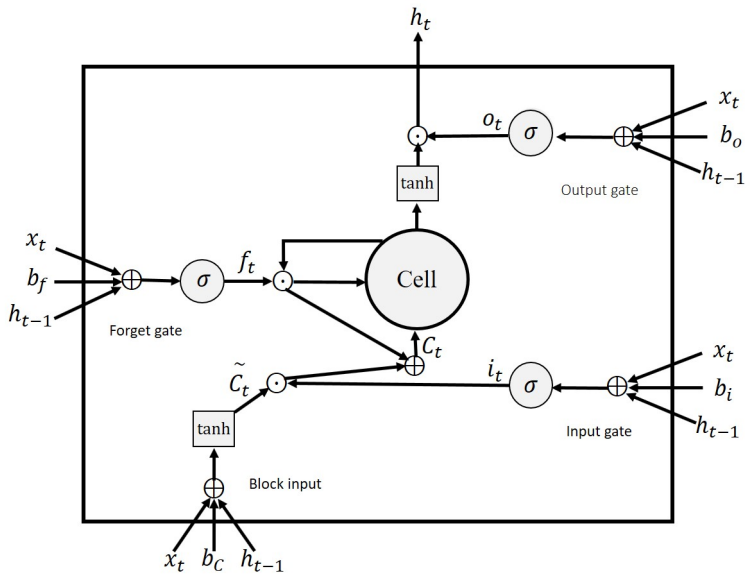

Figure 2.2: An LSTM block with input gate, output gate, forget gate, block input, and sigmoid and hyperbolic tangent activation functions.

is expressed by

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}], \tag{2.9}$$

where $\overrightarrow{h_t}$ is the forward block output and $\overleftarrow{h_t}$ is the backward block output. The final output at time t is

$$y_t = \sigma(W_y h_t + b_y), \tag{2.10}$$

where W_y is the weight matrix, b_y is the bias vector and σ is the activation function. The BLSTM network architecture with forward and backward LSTM layers is shown in Fig. 2.3.

2.1.4 GRU Architecture

A Gated Recurrent Unit (GRU) is a modified LSTM network. The difference is that a GRU has a reset gate and an update gate. A block of a GRU network is shown in Fig. 2.4. In this block, the reset gate is used to decide how much is forgotten. The

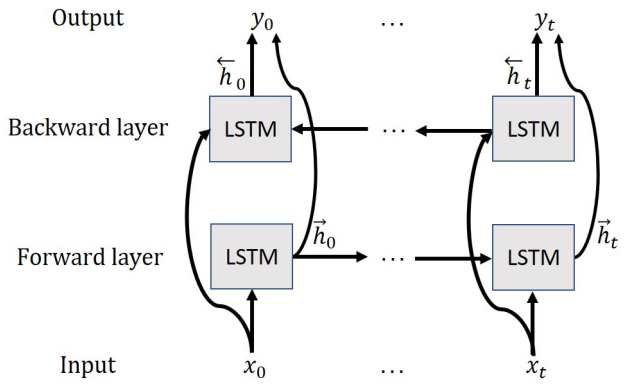

Figure 2.3: BLSTM architecture with forward and backward LSTM layers.

reset gate is expressed by

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \tag{2.11}$$

where W_r and U_r are the weight matrices, and b_r is the bias vector. The update gate is given by

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{2.12}$$

where b_z is the bias, and W_z and U_z are the weight matrices. The block output at time t is then

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{2.13}$$

where W_h and U_h are the weight matrices, and b_h is the bias vector.

2.1.5 Proposed Models

The first proposed model employs two stages, an Autoencoder and LSTM. First, the data set is divided into two sets, positive labeled data (normal) and negative labeled data (abnormal) to train the Autoencoder. The Autoencoder output is then fed into

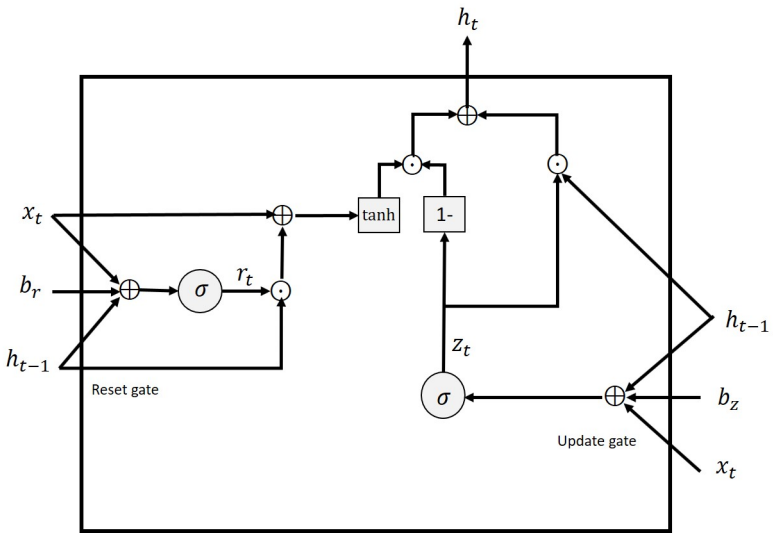

Figure 2.4: A GRU block with reset gate, update gate, sigmoid and hyperbolic tangent activation functions.

the LSTM network for anomaly detection and classification.

The Autoencoder has two networks (positive and negative) with three hidden layers (two encoder and one decoder layers). For the positive Autoencoder network, the positive labeled data is selected and text pre-processing such as removing hyphens and tokenization are applied and the letters changed to lowercase. Next, sentences are truncated to 40 tokens (100 tokens for the IMDB data set) and sentences containing less than 5 tokens are deleted. Then, the number of appearances of each token in the data set is computed and the tokens are ordered from most frequent to least frequent. Each token is given an index starting from zero and the indices are used to replace the tokens in the data set. Then the data set is shuffled. Then an encoder layer with 400 neurons and L1 regularizer is used, followed by an encoder layer with 200 neurons. Next, a decoder layer with 200 neurons is used and finally an output layer with the same input size is applied. This model is trained with just the positive labeled data without the labels. The categorical cross-entropy loss function with Adam optimizer is used to train this model. To prevent overfitting, dropout with probability 0.8

between each layer is employed and early stopping is used. The batch size is 128 and the maximum number of training epochs is 100.

After training, the positive Autoencoder output (which is the same size as the input) is labeled as positive. The negative Autoencoder network has the same architecture as the positive network, the only difference is that negative labeled data is used. After training, the output is labeled as negative. Now the two outputs are concatenated to obtain a single labeled data set for anomaly detection and binary classification. Duplicates are removed and Gaussian noise with zero mean and variance 0.1 is added to avoid overfitting [67].

The LSTM network is now used for anomaly detection and classification. This network has a single hidden layer. First, the concatenated data set is divided into testing and training sets with 95% for testing and 5% for training, and these sets are shuffled. The training set is then divided into two sets with 5% for training and 95% for training validation (except for the IMDB and Openstack data sets with 85% for training and 15% for validation). This is input to a Keras[¶] embedding layer which converts each element to a vector. A hidden LSTM layer of size 100 is used to classify the data into two labels using softmax activation in the final layer. New data are fed to both Autoencoders and the LSTM network to detect anomalies. Categorical cross-entropy is used as the loss function and the Adam optimizer is applied. To prevent overfitting, dropout is used in each LSTM layer with probability 0.8 and early stopping is applied. 10-fold cross-validation is used in training with a maximum of 100 epochs and batch size 128.

This model is called Auto-LSTM. Replacing the LSTM network in this model with the BLSTM and GRU networks gives the Auto-BLSTM and Auto-GRU models, respectively. All three models have the same hyperparameters as described above.

[¶] https://github.com/keras-team/keras

The Auto-GRU architecture with two Autoencoder networks and a GRU network for anomaly detection and classification is shown in Fig. 2.5.

2.2 Results

In this section, the Auto-LSTM, Auto-BLSTM and Auto-GRU models are evaluated using four data sets, namely BGL, IMDB, Openstack and Thunderbird. The following four criteria are used to evaluate the performance: accuracy, precision, recall and F-measure. Accuracy is the fraction of input data that are correctly predicted and is given by

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}, \tag{2.14}$$

where T_p is the number of positive instances predicted by the model to be positive, T_n is the number of negative instances predicted to be negative, F_p is the number of negative instances predicted to be positive and F_n is the number of positive instances predicted to be negative. Precision is given by

$$P = \frac{T_p}{T_p + F_p}, \tag{2.15}$$

and recall is expressed as

$$R = \frac{T_p}{T_p + F_n}. \tag{2.16}$$

The F-measure is the harmonic mean of recall and precision

$$F = \frac{2 \times P \times R}{P + R}. \tag{2.17}$$

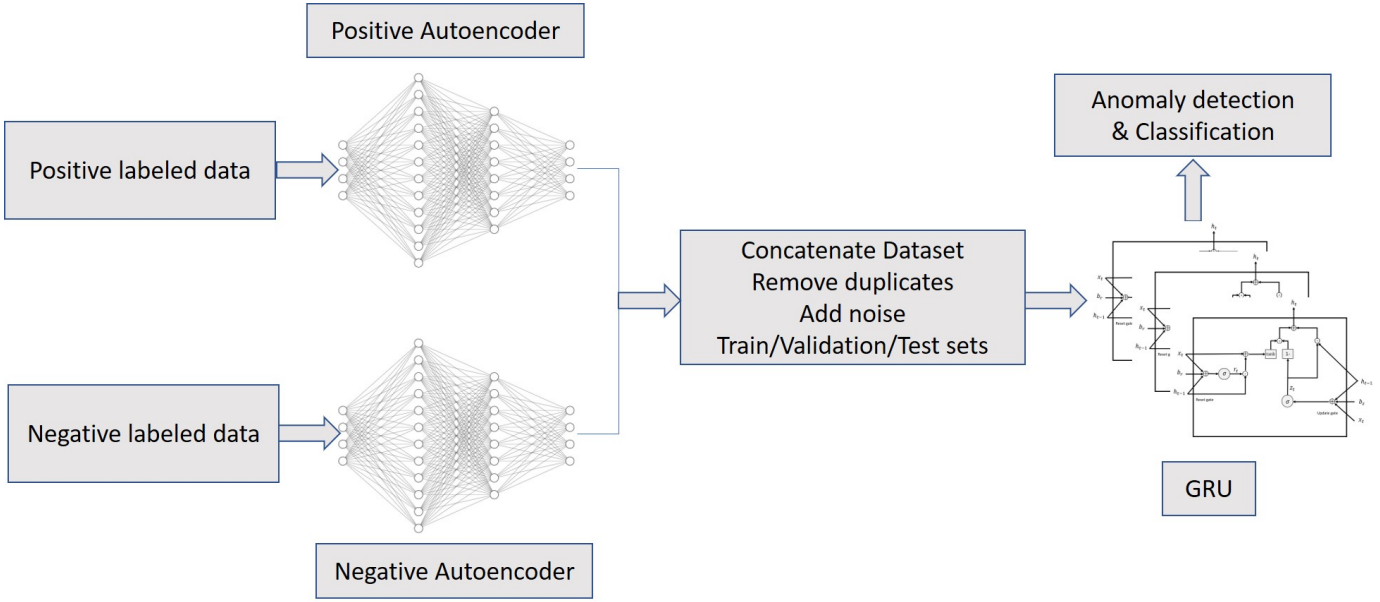

Figure 2.5: The Auto-GRU architecture with two Autoencoder networks and a GRU network for anomaly detection and classification.

All experiments were executed on the Compute Canada Graham cluster with 32 CPU cores, 124 GB memory and two P100 GPUs with Python, Keras and Tensorflow[‖]. The default hyperparameters were used for all data sets so they are not tuned for the proposed models. Tables 2.2 to 2.4 give the results for the BGL, IMDB, Openstack and Thunderbird data sets with the Auto-LSTM, Auto-BLSTM and Auto-GRU models, respectively. For each data set, the average training accuracy, average validation accuracy, average training loss, testing accuracy, precision, recall and F-measure are shown.

2.2.1 BGL

From the BlueGene/L (BGL) data set, 4,399,502 positive logs and 348,460 negative logs were obtained from the Autoencoder. Of these, 11,869 logs were used for training, 225,529 for validation and the remaining 4,510,564 for testing.

With the Auto-LSTM model, the average training accuracy is 98.2% and validation is 98.4% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.09 with a standard deviation of 0.01. The testing accuracy is 99.2% with 98.0% precision for the negative logs and 99.3% for the positive logs, and recall of 91.3% and 99.8% for negative and positive logs, respectively. The F-measure is 94.5% and 99.5% for negative and positive logs, respectively.

With the Auto-BLSTM model, the average training accuracy is 98.4% and validation is 98.7% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.07 with a standard deviation of 0.01. The testing accuracy is 99.3% with precision of 98.9% for negative logs and 99.4% for positive logs, and recall of 92.1% and 99.9% for negative and positive logs, respectively. The F-measure is 95.4% and 99.6% for negative and positive logs, respectively.

[‖] https://github.com/tensorflow/tensorflow

With the Auto-GRU model, the average training accuracy is 97.8% and validation is 98.6% with standard deviations of 0.02 and 0.01, respectively, in 10-fold cross-validation. The average training loss is 0.07 with a standard deviation of 0.01. The testing accuracy is 99.3% with 98.9% precision for negative logs and 99.3% for positive logs, and recall of 91.6% and 99.9% for negative and positive logs, respectively. The F-measure is 95.1% and 99.6% for negative and positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. The precision, recall and F-measure results for negative logs are also better than the 82.5%, 94.7% and 88.2%, respectively, with the nLSA-Log algorithm [98], and the 83%, 99% and 91%, respectively, with SVM unsupervised learning [30] (however, their recall is higher). Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the BGL data set with the Dirichlet Process Gaussian Mixture Model (BGM), Elliptical Envelope (EEnvelope), Gaussian Mixture Model (GMM), K-means, Local Outlier Factor (LOF), and One-Class Support Vector Machine (OC-SVM) algorithms using 10-fold cross-validation are given in Table 2.1(a). The results show that the proposed model is significantly better than these algorithms.

2.2.2 IMDB

The IMDB data set consists of 50,000 movie review sentences, with equal numbers that are positive and negative. The Autoencoder reduced this to 49,565 sentences with 24,875 positive and 24,690 negative. From these, 2,106 sentences were used for training, 372 for validation and the remaining 47,087 for testing.

With the Auto-LSTM model, the average training accuracy and average validation

accuracy are 97.2% and 97.9%, respectively, with standard deviations of 0.01 and 0.02, respectively, in 10-fold cross-validation. The average training loss is 0.08 with a standard deviation of 0.03. The testing accuracy is 97.8% with a precision of 97.9% for negative labels and 97.7% for positive labels, and recall of 97.7% and 97.9% for negative and positive labels, respectively. The F-measure is 97.8% for both labels.

With the Auto-BLSTM model, the average training accuracy and average validation accuracy are 98.3% and 99.1%, respectively, with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with standard deviation 0.01. The testing accuracy is 98.8% with a precision of 99.6% for negative labels and 98.0% for positive labels, and recall of 98.0% and 99.6% for negative and positive labels, respectively. The F-measure is 98.8% and 98.8% for the negative and positive labels, respectively.

With the Auto-GRU model, the average training accuracy and average validation accuracy is 97.5% and 98.4%, respectively, with a standard deviation of 0.02 in 10-fold cross-validation in the training. The average training loss is 0.08 with a standard deviation of 0.04. The testing accuracy is 98.8% with a precision of 99.8% for negative labels and 97.9% for positive labels, and recall of 97.8% and 99.8% for negative and positive labels, respectively. The F-measure is 98.8% and 98.8% for the negative and positive labels, respectively. The accuracy is better than 93.2% with a Convolutional Recurrent Network which is the combination of a Convolutional Neural Network and a Recurrent Neural Network [28]. The accuracy is also better than 94.5% with a Paragraph Vector and 89.1% for a simple LSTM network [34].

2.2.3 Openstack

For the Openstack data set, 137,074 positive log messages and 18,434 negative log messages were obtained from the Autoencoder. From these, 6,608 messages were used

for training, 1,167 for validation and the remaining 147,733 logs for testing.

With the Auto-LSTM model, the average training accuracy is 98.5% and average validation accuracy is 98.6% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with a standard deviation 0.01. The testing accuracy is 99.1% with precision of 99.4% for negative logs and 99.0% for positive logs, and recall of 92.8% and 99.9% for negative and positive logs, respectively. The F-measure is 96.0% and 99.5% for negative and positive logs, respectively.

With the Auto-BLSTM model, the average training accuracy is 98.5% and the average validation accuracy is 99.4% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with a standard deviation of 0.02. The testing accuracy is 99.4% with precision of 99.6% for negative logs and 99.3% for positive logs, and recall of 95.2% and 99.9% for negative and positive logs, respectively. The F-measure is 97.3% and 99.6% for negative and positive logs, respectively.

With the Auto-GRU model, the average training accuracy is 98.4% and the average validation accuracy is 97.2% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with a standard deviation of 0.01. The testing accuracy is 98.3% with precision of 97.9% for negative logs and 98.3% for positive logs, and recall of 87.1% and 99.8% for the negative and positive logs, respectively. The F-measure is 92.2% and 99.0% for negative and positive logs, respectively.

The precision, recall and F-measure results for negative logs are similar to the 94%, 99% and 97% obtained with the Deeplog network [14] (however, their recall is higher). Experiments were also conducted with several well-known anomaly detection algorithms. The average testing accuracy, precision, recall, F-measure and time for the Openstack data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 2.1(b). The results show that the proposed model is significantly better than these algorithms.

2.2.4 Thunderbird

For the Thunderbird data set, 3,000,000 positive log messages and 3,248,239 negative log messages were obtained from the Autoencoder output. From these, 15,620 messages were used for training, 296,791 for validation and the remaining 5,935,828 for testing.

With the Auto-LSTM model, the average training accuracy is 97.3% and the average validation accuracy is 98.9% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.09 with a standard deviation of 0.02. The testing accuracy is 99.0% with precision of 98.4% and 99.8% and recall of 99.8% and 98.2% for negative and positive logs, respectively. The F-measure is 99.1% and 99.0% for negative and positive logs, respectively.

With the Auto-BLSTM model, the average training accuracy is 98.0% and the average validation accuracy is 99.6% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.07 with a standard deviation of 0.04. The testing accuracy is 99.4% with precision of 99.0% and 99.8% and recall of 99.9% and 98.9% for negative and positive logs, respectively. The F-measure is 99.4% and 99.3% for negative and positive logs, respectively.

With the Auto-GRU model, the average training accuracy is 97.5% and the average validation accuracy is 99.3% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.08 with a standard deviation of 0.02. The testing accuracy is 99.2% with precision of 98.7% and 99.9% and recall of 99.9% and 98.5% for negative and positive logs, respectively. The F-measure is 99.3% and 99.2% for negative and positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 96%, 96% and 96%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. Experiments were also conducted with several well-known

algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the Thunderbird data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 2.1(c). The results show that the proposed model is significantly better than these algorithms.

2.2.5 Discussion

The proposed models provided good results for all four data sets. The results for the Auto-BLSTM model were slightly better than for the Auto-LSTM and Auto-GRU models except for the IMDB data set which was the same as with Auto-GRU. Further, Auto-GRU was slightly better than Auto-LSTM except for the Openstack data set. In the proposed models, very little training data was used whereas Deep Learning algorithms typically need significant data for learning. This is because of the feature extraction using Autoencoders and the pre-processing. Although the focus here was on log anomaly detection and classification, the proposed models were evaluated using the IMDB data set to show that they can also provide good results on other text classification tasks such as sentiment classification. Hyperparameter tuning such as the learning rate, network structure and the number of hidden layers should improve the results obtained.

Training the Autoencoder network is the most important stage. In the proposed models, the original data is converted from text to digits using the word frequency. Then the Autoencoder is trained and information extracted using this converted data as input which provides a better relationship with the original data for Machine Learning algorithms. These proposed models are particularly suited to log messages because these messages contain unstructured data which can be challenging for anomaly detection and classification algorithms.

2.3 Conclusion

Anomaly detection and classification of log messages is a very important task in Machine Learning. In this chapter, the Auto-LSTM, Auto-BLSTM and Auto-GRU models were proposed for this purpose. The first stage of these models employs an Autoencoder network to extract features from the input data and the second stage is anomaly detection and classification with an LSTM, BLSTM or GRU network. The proposed models were tested on four different data sets. Three of these data sets are log messages for anomaly detection and classification while the fourth is a sentiment movie review classification data set. The models were shown to provide good results for these data sets with only a very small portion used for training. This indicates that these models can be used for tasks other than log message anomaly detection and classification such as text classification. However, the proposed models results were good, the models were affected by the imbalanced logs. In the next chapter, a model is proposed with oversampling log messages to improve the performance.

Table 2.1: The average testing accuracy, precision, recall, F-measure and time for the BGM, EEnvelope, GMM, K-means, LOF and OC-SVM algorithms using 10-fold cross-validation with the (a) BGL, (b) Openstack, and (c) Thunderbird data sets. Positive labels are denoted by 1 and negative labels by 0.

	Algorithm	Testing Accuracy	Label	Precision	Recall	F-measure	Time (s)
(a)	**BGM**	59.4%	0	42.8%	60.0%	50.0%	3892
			1	71.7%	59.4%	61.4%	
	EEnvelope	85.2%	0	12.8%	17.5%	14.8%	6197
			1	93.3%	90.6%	91.9%	
	GMM	68.0%	0	46.0%	61.8%	52.5%	1928
			1	76.8%	68.5%	69.9%	
	K-means	48.6%	0	1.3%	10.0%	2.3%	18571
			1	88.1%	51.7%	65.1%	
	LOF	83.5%	0	7.0%	10.2%	8.3%	1462
			1	92.6%	89.3%	90.9%	
	OC-SVM	84.3%	0	8.4%	11.4%	9.7%	56818
			1	92.7%	90.1%	91.4%	
(b)	**BGM**	57.5%	0	31.6%	60.0%	37.1%	95
			1	77.0%	57.2%	63.3%	
	EEnvelope	80.8%	0	16.8%	15.6%	16.2%	318
			1	88.8%	89.6%	89.2%	
	GMM	58.0%	0	37.6%	50.0%	40.3%	81
			1	71.3%	59.1%	62.5%	
	K-means	40.0%	0	40.0%	40.0%	40.0%	378
			1	40.0%	40.0%	40.0%	
	LOF	80.2%	0	14.2%	13.2%	13.7%	1541
			1	88.4%	89.3%	88.8%	
	OC-SVM	38.5%	0	0.3%	1.3%	0.5%	41280
			1	76.6%	43.5%	55.5%	
(c)	**BGM**	67.0%	0	21.4%	40.1%	26.8%	3203
			1	84.2%	72.4%	74.3%	
	EEnvelope	75.4%	0	10.8%	6.5%	8.1%	5027
			1	82.6%	89.3%	85.8%	
	GMM	60.0%	0	25.9%	40.1%	31.5%	987
			1	74.9%	64.0%	67.4%	
	K-means	68.8%	0	13.8%	16.7%	15.1%	11976
			1	82.6%	79.2%	80.9%	
	LOF	75.5%	0	17.6%	10.2%	12.9%	1306
			1	82.2%	89.6%	85.7%	
	OC-SVM	40.7%	0	8.4%	23.7%	12.5%	50926
			1	72.9%	44.3%	55.1%	

Table 2.2: Auto-LSTM results for the BGL, IMDB, Openstack and Thunderbird data sets (the numbers in parenthesis are standard deviation). Positive labels are denoted by 1 and negative labels by 0.

Data set	**Average Training Accuracy**	**Average Validation Accuracy**	**Average Training Loss**	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**
BGL	98.2% (0.01)	98.4% (0.01)	0.09 (0.01)	99.2%	0	98.0%	91.3%	94.5%
					1	99.3%	99.8%	99.5%
IMDB	97.2% (0.01)	97.9% (0.02)	0.08 (0.03)	97.8%	0	97.9%	97.7%	97.8%
					1	97.7%	97.9%	97.8%
Openstack	98.5% (0.01)	98.6% (0.01)	0.05 (0.01)	99.1%	0	99.4%	92.8%	96.0%
					1	99.0%	99.9%	99.5%
Thunderbird	97.3% (0.01)	98.9% (0.01)	0.09 (0.02)	99.0%	0	98.4%	99.8%	99.1%
					1	99.8%	98.2%	99.0%

Table 2.3: Auto-BLSTM results for the BGL, IMDB, Openstack and Thunderbird data sets (the numbers in parenthesis are standard deviation). Positive labels are denoted by 1 and negative labels by 0.

Data set	**Average Training Accuracy**	**Average Validation Accuracy**	**Average Training Loss**	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**
BGL	98.4% (0.01)	98.7% (0.01)	0.07 (0.01)	99.3%	0	98.9%	92.1%	95.4%
					1	99.4%	99.9%	99.6%
IMDB	98.3% (0.01)	99.1% (0.01)	0.05 (0.01)	98.8%	0	99.6%	98.0%	98.8%
					1	98.0%	99.6%	98.8%
Openstack	98.5% (0.01)	99.4% (0.01)	0.05 (0.02)	99.4%	0	99.6%	95.2%	97.3%
					1	99.3%	99.9%	99.6%
Thunderbird	98.0% (0.01)	99.6% (0.01)	0.07 (0.04)	99.4%	0	99.0%	99.9%	99.4%
					1	99.8%	98.9%	99.3%

Table 2.4: Auto-GRU results for the BGL, IMDB, Openstack and Thunderbird data sets (the numbers in parenthesis are standard deviation). Positive labels are denoted by 1 and negative labels by 0.

Data set	**Average Training Accuracy**	**Average Validation Accuracy**	**Average Training Loss**	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**
BGL	97.8% (0.02)	98.6% (0.01)	0.07 (0.01)	99.3%	0	98.9%	91.6%	95.1%
					1	99.3%	99.9%	99.6%
IMDB	97.5% (0.02)	98.4% (0.02)	0.08 (0.04)	98.8%	0	99.8%	97.8%	98.8%
					1	97.9%	99.8%	98.8%
Openstack	98.4% (0.01)	97.2% (0.01)	0.05 (0.01)	98.3%	0	97.9%	87.1%	92.2%
					1	98.3%	99.8%	99.0%
Thunderbird	97.5% (0.01)	99.3% (0.01)	0.08 (0.02)	99.2%	0	98.7%	99.9%	99.3%
					1	99.9%	98.5%	99.2%

Chapter 3

Oversampling Log Messages Using a Sequence Generative Adversarial Network for Anomaly Detection

In this chapter, a model is proposed to deal with imbalanced log data by oversampling text log messages using a Sequence Generative Adversarial Network (SeqGAN) [101]. The resulting data is then used for anomaly detection with Autoencoder [75] and Gated Recurrent Unit (GRU) [10] networks. The proposed model is evaluated using two labeled log message data sets, namely BlueGene/L (BGL) and Openstack. Results are presented which show that the proposed model with oversampling provides better results than the model without oversampling.

The rest of the chapter is organized as follows. In Section 3.1 the SeqGAN architecture is presented and the proposed model is described. The experimental results and discussion are given in Section 3.2. Finally, Section 3.3 provides some concluding remarks.

3.1 System Model

In this section, the SeqGAN architecture employed is given along with the proposed network model.

3.1.1 SeqGAN Architecture

A SeqGAN consists of a Generator (G) and a Discriminator (D). The Discriminator is trained to discriminate between real data (sentences) and generated sentences. The Generator is trained using the Discriminator using the reward function with policy gradient [82]. In SeqGAN, the reward for a sentence is computed and the Generator is regulated using the reward with reinforcement learning. Generator G_θ is trained with a real data set to produce a sentence

$$Y_{1:T} = \{y_1, \ldots, y_t, \ldots, y_T\}, y_t \in \mathcal{Y},$$

where $\mathcal{Y}$ is the vocabulary of candidate words. This should produce a sentence that is close to real data. This is a Reinforcement Learning problem which considers G_θ to produce an action a (next word y_t) given the state s (previously generated words $Y_{1:t-1}$). SeqGAN trains the Discriminator D_ϕ as well as Generator G_θ. D_ϕ is trained to discriminate between real data and data generated from G_θ. Words are generated by G_θ each time step but D_ϕ only computes rewards for full sentences. Hence, the rewards for intermediate states are estimated using MC search and are given by

$$Q_{D_\phi}^{G_\theta} = (s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC(Y_{1:t}; N) & if: t < T, \\ D_\phi(Y_{1:t}) & if: t = T, \end{cases} \tag{3.1}$$

where $Q_{D_\phi}^{G_\theta}$ is the action-value function which is the expected reward from the Discriminator, T is the sentence length and N is the number of the sentences in the MC search, $Y_{1:T}^n$ is the nth sentence in the MC search, and $D_\phi(Y_{1:T}^n)$ is the probability of the nth sentence being denoted real by the Discriminator.

After the reward is computed, the Generator G_θ is updated via the policy gradient which is the gradient of the objective function and is given by

$$\begin{aligned}\nabla_\theta J(\theta) &\simeq \frac{1}{T}\sum_{t=1}^{T}\sum_{y_t\in\mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \\ &= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{y_t\sim G_\theta(y_t|Y_{1:t-1})}[\nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)], \quad (3.2)\end{aligned}$$

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta), \quad (3.3)$$

where α is the learning rate. SeqGAN updates the Discriminator and Generator until the stopping criteria are satisfied. An LSTM, GRU or other RNN network for the Generator and a Convolutional Neural Network (CNN) network for the Discriminator have been shown to provide good results for classification tasks [101].

The SeqGAN architecture is shown in Fig. 3.1. The orange circles denote words in real sentences and the blue circles denote words in generated sentences. First, the Generator is pre-trained with real data using the cross-entropy loss function which minimizes the negative log-likelihood. Then it is used to generate data and the Discriminator is pre-trained with both generated and real data. Then the MC search parameters (β) are set the same as the Generator parameters (θ). As shown on the right, an MC search is used to compute the reward for an intermediate state. This search generates N complete sentences from the current state. A reward is computed for each sentence and averaged as the intermediate reward except in the last time step where the reward is obtained from the Discriminator. The input of each time

Algorithm 1 The SeqGAN algorithm

Pre-train Generator G_θ with real data
$\beta \leftarrow \theta$
Pre-train Discriminator D_ϕ using the data generated by the Generator and real data
repeat
    for g-steps **do**
        Generate a sentence $Y_{1:T} = \{y_1, \ldots, y_T\} \sim G_\theta$
        for t in $1 : T$ **do**
            Compute the rewards using (3.1)
        Update the Generator parameters with the policy gradient using (3.3)
    for d-steps **do**
        Use the Generator to generate data
        Train the Discriminator with real and generated data
    $\beta \leftarrow \theta$
until stopping criteria are satisfied

step is the output of the previous time step and the next word is obtained via a multinomial distribution over the softmax of the GRU output. Then the Generator is trained with the policy gradient. Finally, the updated Generator is used to generate data and the Discriminator is trained with both the generated and real data. The SeqGAN algorithm is given in Algorithm 1.

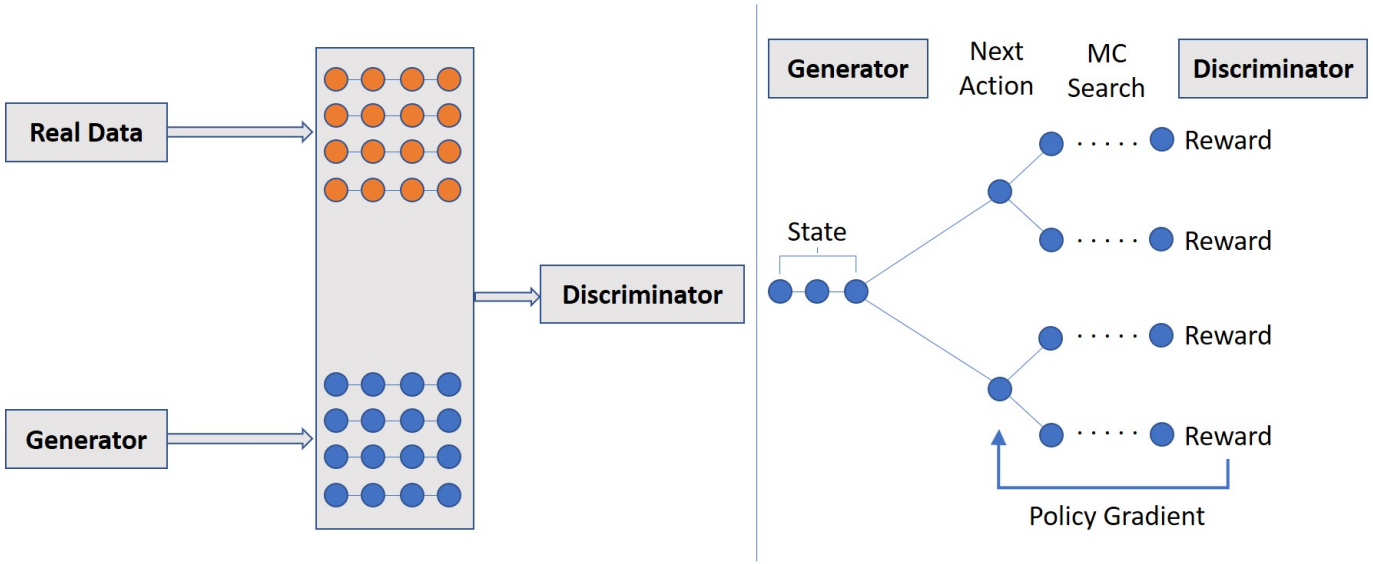

Figure 3.1: The SeqGAN architecture [101]. The Discriminator on the left is trained with real data and data generated using the Generator. The Generator on the right is trained using the policy gradient. The reward is computed by the Discriminator. This is used to determine the intermediate action values for the MC search.

3.1.2 Proposed Model

The proposed model has three stages. The first is generating log messages using SeqGAN for oversampling. The message logs are divided into two data sets, positive labeled data (normal) and negative labeled data (abnormal). Additional negative labeled data is generated using the negative labeled data set. The initial negative data set is split into several sets (the Openstack data set is split into two sets and the BGL data set is split into seven sets), and fed into the SeqGAN separately. This ensures better convergence and provides different negative log messages. Further, the network speed is faster which is important with data generating. A CNN is used in the SeqGAN for the discriminator and a GRU as the generator. The GRU has one hidden layer of size 30 with Adam optimizer and the batch size is 128. The generated negative log messages are concatenated with the original negative data and similar messages are removed. The resulting data set is balanced with similar numbers of positive and negative data.

The second stage is the Autoencoder which has two networks (positive and negative) with three hidden layers (two encoder layers and one decoder layer). The encoder layers have 400 (with L1 regularizer) and 200 neurons and the decoder layer has 200 neurons. The output layer has 40 neurons which is the same size as the input layer. The positive labeled data is fed into the positive Autoencoder. Note that this network is trained with just positive label data. The maximum number of epochs is 100 and the batch size is 128. Dropout with probability 0.8 and early stopping is used to prevent overfitting. Categorical cross-entropy loss with the Adam optimizer is used for training. The network output is labeled as positive. The negative labeled data which has been oversampled is fed into the negative Autoencoder and the network output is labeled as negative. The two sets of labeled data are then concatenated, duplicates are removed and Gaussian noise with zero mean and 0.1 variance is added

to avoid overfitting [67].

The final stage is the GRU network for anomaly detection and classification. First, the concatenated data set is divided into training and testing sets with 5% for training and 95% for testing, and these sets are shuffled. The training set is then divided into two sets with 5% for training and 95% for validation. The data is fed into the GRU hidden layer of size 100 and is classified using softmax activation. 10-fold cross-validation is used in training with a maximum of 100 epochs and a batch size of 128. Dropout with probability 0.8 and early stopping is used to prevent overfitting. Categorical cross-entropy loss with the Adam optimizer is used for training. The proposed model is shown in Fig. 3.2.

3.2 Results

In this section, the proposed model is evaluated with and without SeqGAN oversampling using the BGL and Openstack data sets. The following four criteria are used to evaluate the performance: accuracy, precision, recall and F-measure.

All experiments were conducted on the Compute Canada Cedar cluster with 24 CPU cores, 125 GB memory and four P100 GPUs with Python, Keras and Tensorflow. The hyperparameters of the proposed model are not tuned so the default values were used for all data sets. For each data set, the average training accuracy, average validation accuracy, average training loss, testing accuracy, precision, recall and F-measure were obtained. Tables 3.1 and 3.2 give the results for the BGL and Openstack data sets without and with SeqGAN oversampling, respectively.

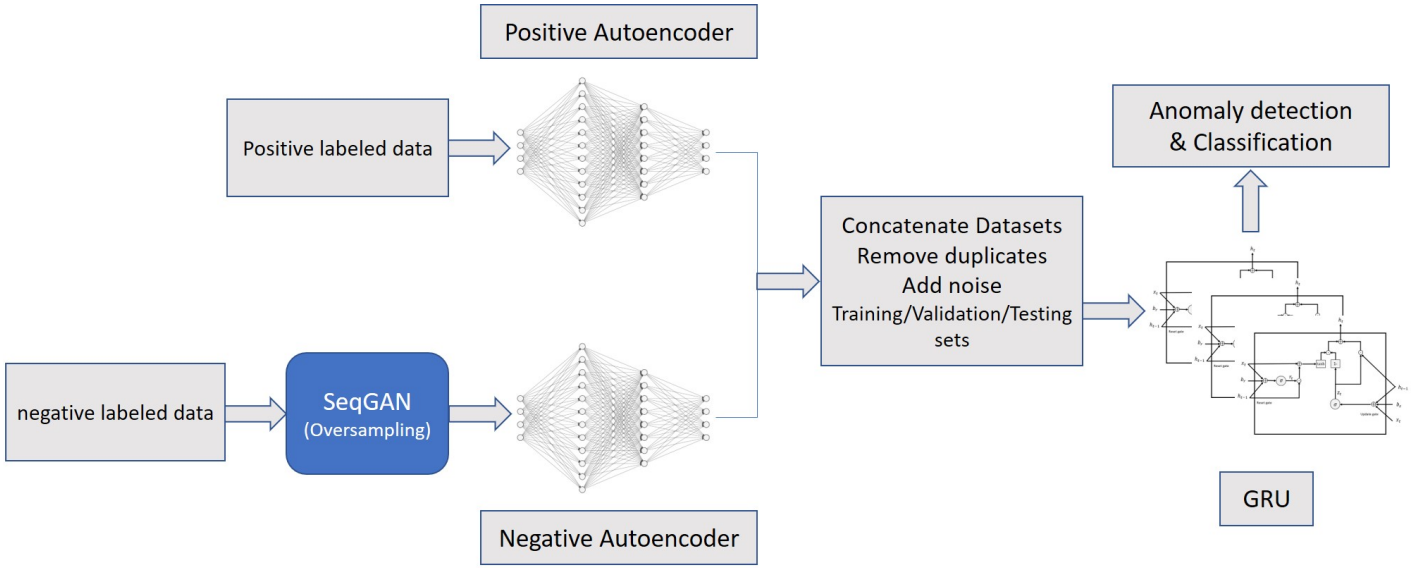

Figure 3.2: The proposed model architecture with SeqGAN for oversampling log messages, two Autoencoder networks and a GRU network for anomaly detection and classification.

3.2.1 BGL

The BlueGene/L (BGL) data set consists of 4,399,502 positive log messages and 348,460 negative log messages (without oversampling). From this data set, 11,869 logs are used for training, 225,529 for validation and the remaining 4,510,564 for testing. Without oversampling, the average training accuracy is 97.8% and average validation accuracy is 98.6% with standard deviations of 0.02 and 0.01, respectively, in 10-fold cross-validation. The average training loss is 0.07 with a standard deviation of 0.01. The testing accuracy is 99.3% with a precision of 98.9% for negative logs and 99.3% for positive logs, and recall of 91.6% and 99.9% for negative and positive logs, respectively. The F-measure is 95.1% and 99.6% for negative and positive logs, respectively.

Oversampling of the negative log messages with SeqGAN increased the number in the BGL data set to 4,137,516 so the numbers of positive and negative log messages are similar. From this data set, 21,342 logs are used for training, 405,508 for validation and the remaining 8,110,168 for testing with similar numbers of positive and negative log messages in each group. The average training accuracy is 98.3% and average validation accuracy is 99.3% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with a standard deviation of 0.01. The testing accuracy is 99.6% with a precision of 99.8% for negative logs and 99.4% for positive logs, and recall of 99.3% and 99.8% for negative and positive logs, respectively. The F-measure is 99.6% for both negative and positive logs.

The precision, recall and F-measure for negative logs with oversampling are better than the values of 83%, 99% and 91%, respectively, with SVM unsupervised learning [30]. The precision, recall and F-measure results for negative logs are also better than the 82.5%, 94.7% and 88.2%, respectively, with the nLSALog algorithm [98]. The precision, recall and F-measure results for negative logs are also better than the

92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the BGL data set with the Dirichlet Process Gaussian Mixture Model (BGM), Elliptical Envelope (EEnvelope), Gaussian Mixture Model (GMM), K-means, Local Outlier Factor (LOF), and One-Class Support Vector Machine (OC-SVM) algorithms using 10-fold cross-validation are given in Table 2.1(a). The results show that the proposed model is significantly better than these algorithms.

3.2.2 Openstack

The Openstack data set without oversampling consists of 137,074 positive log messages and 18,434 negative log messages. From this data set, 6,608 logs are used for training, 1,167 for validation and the remaining 147,733 for testing. Without oversampling, the average training accuracy is 98.4% and average validation accuracy is 97.2% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.05 with a standard deviation of 0.01. The testing accuracy is 98.3% with a precision of 97.9% for negative logs and 98.3% for positive logs, and recall of 87.1% and 99.8% for negative and positive logs, respectively. The F-measure is 92.2% and 99.0% for negative and positive logs, respectively.

Oversampling of the negative log messages with SeqGAN increased the number in the Openstack data set to 154,202. From this data set, 12,378 logs are used for training, 2,185 for validation and the remaining 276,713 for testing with similar numbers of positive and negative log messages in each group. With oversampling, the average training accuracy is 98.0% and average validation accuracy is 98.7% with a standard deviation of 0.01 in 10-fold cross-validation. The average training loss is 0.06 with a standard deviation of 0.01. The testing accuracy is 98.9% with a precision

of 99.6% for negative logs and 98.2% for positive logs, and recall of 98.4% and 99.5% for the negative and positive logs, respectively. The F-measure is 99.0% and 98.8% for negative and positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 94%, 99% and 97% obtained with the Deeplog network [14]. Experiments were also conducted with several well-known anomaly detection algorithms. The average testing accuracy, precision, recall, F-measure and time for the Openstack data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 2.1(b). The results show that the proposed model is significantly better than these algorithms.

3.2.3 Discussion

The proposed oversampling with SeqGAN provided good results for both the BGL and Openstack data sets. It is evident that oversampling significantly improved the model accuracy for negative log messages. For the BGL data set, the precision, recall and F-measure after oversampling increased from 98.9% to 99.8%, 91.6% to 99.3% and 95.1% to 99.6% which are 0.9%, 7.7% and 4.5% higher, respectively. For the Openstack data set, the precision, recall and F-measure after oversampling increased from 97.9% to 99.6%, 87.1% to 98.4% and 92.2% to 99.0% which are 1.7%, 11.3% and 6.8% higher, respectively. These results show that data balancing should be considered with Deep Learning algorithms to improve accuracy, especially for small numbers of minor label samples. The proposed model was evaluated with two data sets for anomaly detection and classification with only a small portion used for training. This is an important result because DL algorithms typically require significant amounts of data for training. Note that good results were obtained even though the hyperparameters were not tuned.

The first stage in the proposed model where logs are oversampled with a SeqGAN network is the most important. These networks have been shown to provide promising results in generating text such as poems [96]. The concept of generating data is similar to that for oversampling. The second stage which extracts features from the data using an Autoencoder is also important. The Autoencoder output is very suitable for use with an RNN based algorithm such as a GRU for anomaly detection and classification. The results obtained show that the proposed model can provide excellent results even when the data is imbalanced.

3.3 Conclusion

In this chapter, a model was proposed to address the problem of imbalanced log messages. In the first stage, the negative logs were oversampled with a SeqGAN network so that the numbers of positive and negative logs are similar. The resulting labeled logs were then fed into an Autoencoder to extract features from the text data. Finally, a GRU network was used for anomaly detection and classification. The proposed model was evaluated using two log message data sets, namely BGL and Openstack. Results were presented which show that oversampling can improve detection and classification accuracy. Although the supervised model results were good, it can be difficult to obtain high volume of labeled logs for normal and abnormal system operation. In the next chapter, an unsupervised model is proposed using Isolation Forest and Autoencoder networks to address this problem.

Table 3.1: Results without oversampling for the BGL and Openstack data sets (numbers in parenthesis are standard deviation). Positive labels are denoted by 1 and negative labels by 0.

Data set	**Average Training Accuracy**	**Average Validation Accuracy**	**Average Training Loss**	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**
BGL	97.8% (0.02)	98.6% (0.01)	0.07 (0.01)	99.3%	0	98.9%	91.6%	95.1%
					1	99.3%	99.9%	99.6%
Openstack	98.4% (0.01)	97.2% (0.01)	0.05 (0.01)	98.3%	0	97.9%	87.1%	92.2%
					1	98.3%	99.8%	99.0%

Table 3.2: Results with oversampling using SeqGAN for the BGL and Openstack data sets (numbers in parenthesis are standard deviation). Positive labels are denoted by 1 and negative labels by 0.

Data set	**Average Training Accuracy**	**Average Validation Accuracy**	**Average Training Loss**	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**
BGL	98.3% (0.01)	99.3% (0.01)	0.05 (0.01)	99.6%	0	99.8%	99.3%	99.6%
					1	99.4%	99.8%	99.6%
Openstack	98.0% (0.01)	98.7% (0.01)	0.06 (0.01)	98.9%	0	99.6%	98.4%	99.0%
					1	98.2%	99.5%	98.8%

Chapter 4

Unsupervised Log Message Anomaly Detection with Isolation Forest and Autoencoder

Isolation Forest [50] has been used to find abnormal data, but in this chapter it is used to detect normal data with a threshold. An Autoencoder [75] is used to extract features for Isolation Forest. The proposed architecture includes two Autoencoder networks and Isolation Forest. Feature extraction using Autoencoders has two main advantages. First, it improves the Isolation Forest results as Isolation Forest alone does not provide good results for log message anomaly detection. Further, Isolation Forest is used to predict positive data instead of negative data and use it as a threshold. Second, an Autoencoder is able to separate anomalies from normal data very well. Two Autoencoders are used to provide better separation. The proposed model is evaluated using the accuracy, precision, recall and F-measure metrics with three log message data sets, namely BlueGene/L (BGL), Openstack and Thunderbird.

The rest of this chapter is organized as follows. In Section 4.1, the Isolation Forest architecture is presented and the proposed model is described. The results and

discussion for the three data sets are given in Section 4.2. Finally, some concluding remarks are given in Section 4.3.

4.1 System Model

In this section, the Isolation Forest architecture is given along with the proposed model.

4.1.1 Isolation Forest

Isolation Forest [50] is an ensemble approach which has been used to detect anomalies. It begins with a random attribute and chooses a partition between the lowest and highest values in order to separate a sample. This continues until the samples are isolated. The Isolation Forest is built by adding a number of Isolation Trees separated into different attributes. The number of partitions needed to isolate a sample is equal to the path length passed from the root to a leaf.

The Isolation Tree technique is based on Extra Trees [18]. In Isolation Tree, each division is random. A sample located near the root, i.e. its path is short, implies that it is easier to distinguish and therefore easier to isolate from samples that are in deeper leaves. It is expected that anomalies (abnormal samples) will have a smaller average path length than positive (normal) samples. When a sample is at a leaf deep in the tree, the score will be low (close to 0), while if it is shallow the score will be high (close to 1). The anomaly score of sample x for an Isolation Forest F is

$$s(x, N) = 2^{\frac{-E(h(x))}{c(N)}}, \tag{4.1}$$

where N is the number of samples 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 which

is [51]

$$c(N) = \begin{cases} 2H(N-1) - 2(N-1)/N & \text{if} \quad N > 2, \\ 1 & \text{if} \quad N = 2, \\ 0 & \text{otherwise}, \end{cases} \tag{4.2}$$

where $H(N-1)$ is the harmonic number given by $\ln(N-1) + 0.5772156649$.

4.1.2 Proposed Model

The proposed model for unsupervised anomaly detection includes two deep Autoencoder networks and an Isolation Forest. First, text pre-processing including tokenization and changing letters to lowercase are applied to the data set. Next, the sentences are padded to 40 tokens, and sentences including less than five tokens are eliminated. Then, the number of appearances of each token in the data set is computed and the tokens are ordered from most frequent to least frequent. Each token is given an index starting from zero and the indices are used to replace the tokens in the data set. Next, the data set is normalized so all values are between 0 and 1 and the entries are shuffled. Then it is divided into the first training set t_1 (around 0.5% for BGL and Thunderbird, and 5% for Openstack), second training set t_2 (around 2% for BGL and Thunderbird, and 17% for Openstack) and testing set t_3 (around 97.5% for BGL and Thunderbird, and 78% for Openstack). The proportion of positive and negative logs in these sets is the same as in the original data sets. The sets t_1 and t_2 are small as their size affects the speed of the algorithm. The BGL and Thunderbird data sets are larger than Openstack so a smaller proportion of the data is used for these sets. Further, the Openstack data set is small so a larger proportion of the data is needed for training convergence.

The first Autoencoder is used for feature extraction from a small amount of data in an unsupervised way. The first Autoencoder is trained with t_1 (which contains both

positive and negative log messages). This Autoencoder has an encoder layer with 512 units and L1 regularizer, followed by a decoder layer with 256 units. The output layer has the same size as the input which is 40 units. The categorical cross-entropy loss function, Relu activation function and Adam optimizer are used to train this model with a batch size of 64 and a maximum of 500 training epochs. The Adam optimizer is used because it has fast convergence and good performance in DL algorithms [81]. Early stopping is used to prevent overfitting. After training the first Autoencoder, the second training set t_2 and testing set t_3 are fed into this Autoencoder to extract features (as the last layer output which has the same size as the input), which are denoted as f_2 and f_3, respectively.

Feature set f_2 is fed into the Isolation Forest with 100 Isolation Trees and this is tested using f_3 to predict the data. Next, a percentage of the positive predicted data (30% for BGL and Thunderbird, and 70% for Openstack), is randomly chosen from the Isolation Forest output and this is denoted as p_1. The remaining positive predicted data and the negative predicted data are denoted as p_2. p_1 is used to train the second Autoencoder which has the same architecture as the first Autoencoder. The maximum number of training epochs is 100 and early stopping is used to prevent overfitting. After training the second Autoencoder, p_1 is input to this Autoencoder to extract features (as the last layer output of the trained Autoencoder), which is denoted as o_1, and then p_2 is input to this Autoencoder and the extracted features are denoted as o_2. Anomalies are detected using a threshold. To determine the threshold, the average and standard deviation of the feature values of each sample in o_1 are computed. The threshold is given by

$$T = stdv \times c, \tag{4.3}$$

where $stdv$ is the standard deviation of the positive data predicted with Isolation

Forest (o_1) and c is a constant. For unlabeled data, this constant can be estimated based on the final predicted test data. In the data sets, it is known that about 10% of the data is negative and the remainder positive. Thus, the constant can be chosen based on the percentages obtained. The larger data sets (BGL and Thunderbird) were divided into smaller size sets and used in the proposed model. It was found that smaller data sets require a larger constant. Based on the results, the constants for BGL, Thunderbird and Openstack are $c = 0.5$, 0.1 and 5, respectively.

The second Autoencoder needs fewer epochs to train because the features have been extracted with the first Autoencoder. Further, there is more input data for the second Autoencoder than the first Autoencoder, and typically Deep Learning algorithms need large amounts of data to work well. The loss and accuracy of the model were used to determine the number of epochs. The proposed model algorithm is given in Algorithm 2 and the architecture of the proposed model is given in Fig. 4.1.

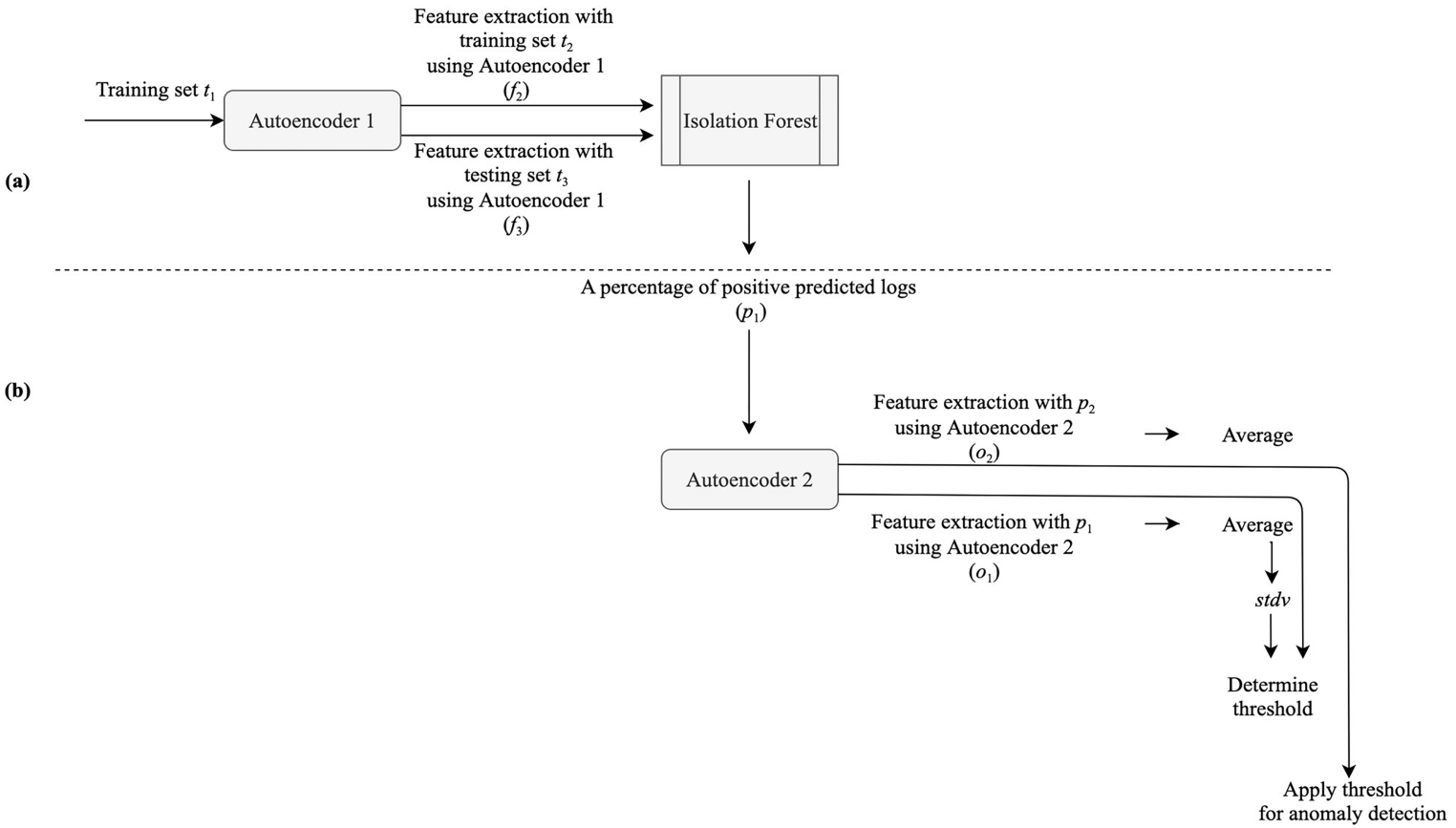

Figure 4.1: Architecture of the proposed model: (a) Isolation Forest with one Autoencoder, and (b) the second Autoencoder for anomaly detection (Isolation Forest with two Autoencoders).

Algorithm 2 Proposed Model Algorithm

Input: first training set: t_1; second training set: t_2; testing set: t_3.

1: **for** epochs **do**
2: Train the first Autoencoder with t_1.
3: Compute f_2 and f_3 using t_2 and t_3, respectively.
4: Train Isolation Forest with f_2 and test with f_3.
5: Randomly select a percentage of the positive predicted logs as p_1, and the remaining predicted logs are p_2.
6: **for** epochs **do**
7: Train the second Autoencoder with p_1.
8: Compute o_1 and o_2 using p_1 and p_2, respectively.
9: Compute the average and standard deviation of the feature values of each sample in o_1 and determine the threshold using (4.3).
10: Compute the average of the feature values of each sample in o_2.
11: If an average is less than the threshold, flag the sample as an anomaly.

4.2 Results

In this section, the proposed model is evaluated using the BGL, Openstack and Thunderbird data sets. Four criteria, namely accuracy, precision, recall and F-measure, are used to evaluate the performance. All experiments were run on the Compute Canada Cedar cluster with 24 CPU cores, four P100 GPUs and 125 GB of memory, and the algorithms were implemented using Python, Keras and Scikit-learn*.

The hyperparameters of the proposed model were not tuned so for all data sets the default values were used. Each experiment was repeated 10 times, and the minimum, maximum and average testing accuracy, precision, recall, F-measure and time were obtained. Table 4.1 gives the results for the BGL, Openstack and Thunderbird data sets for (a) Isolation Forest, (b) Isolation Forest with one Autoencoder, and (c) Isolation Forest with two Autoencoders (proposed model). For Isolation Forest alone, the first training set t_1 is input to the Isolation Forest and the results are obtained using the testing set t_3. For Isolation Forest with one Autoencoder, the first training

*https://github.com/scikit-learn/scikit-learn

set t_1 is input to the first Autoencoder for training. Then the second training set t_2 and testing set t_3 are input to this trained Autoencoder for feature extraction, giving f_2 and f_3, respectively. Then f_2 is fed into the Isolation Forest for training and the results are obtained using f_3. Isolation Forest with two Autoencoders indicates that the results are obtained with the second Autoencoder using a threshold (proposed model).

4.2.1 BGL

The BlueGene/L (BGL) data set has 4,399,502 positive logs and 348,460 negative logs. From these, 23,997 logs are used for the first training set, 93,551 for the second training and the remaining 4,630,414 for testing. With the Isolation Forest model, the average testing accuracy is 88.7% with average precision, recall and F-measure of 28.9%, 36.6% and 32.3% for negative logs, and 94.8%, 92.8% and 93.8% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 90.8% with average precision, recall and F-measure of 42.7%, 76.5% and 54.3% for negative logs, and 98.0%, 91.9% and 94.9% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.6% with average precision, recall and F-measure of 96.8%, 98.7% and 98.1% for negative logs, and 99.8%, 99.7% and 99.8% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 82.5%, 94.7% and 88.2%, respectively, with the nLSALog algorithm [98], and the 83%, 99% and 91%, respectively, with SVM unsupervised learning [30]. The precision, recall and F-measure results for negative logs are also better than the 92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and

time for the BGL data set with the Dirichlet Process Gaussian Mixture Model (BGM), Elliptical Envelope (EEnvelope), Gaussian Mixture Model (GMM), K-means, Local Outlier Factor (LOF), and One-Class Support Vector Machine (OC-SVM) algorithms using 10-fold cross-validation are given in Table 2.1(a). The results show that the proposed model is significantly better than these algorithms.

4.2.2 Openstack

The Openstack data set has 137,074 positive log messages and 18,434 negative log messages. From these, 7,353 logs are used for first training set, 26,545 for the second training and the remaining 121,610 for testing. With the Isolation Forest model, the average testing accuracy is 86.9% with average precision, recall and F-measure of 59.2%, 57.6% and 58.4% for negative logs, and 92.9%, 93.4% and 93.1% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 92.9% with average precision, recall and F-measure of 67.1%, 99.5% and 80.2% for negative logs, and 99.7%, 91.8% and 95.6% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.1% with average precision, recall and F-measure of 96.1%, 97.5% and 96.8% for negative logs, and 99.6%, 99.3% and 99.4% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are similar to the 94%, 99% and 97% obtained with the Deeplog network [14]. Experiments were also conducted with several well-known anomaly detection algorithms. The average testing accuracy, precision, recall, F-measure and time for the Openstack data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 2.1(b). The results show that the proposed model is significantly better than these algorithms.

4.2.3 Thunderbird

From the Thunderbird data set 3,000,000 positive log messages and 600,000 negative log messages are used. From these, 17,000 messages are used for the first training set, 65,700 for the second training and the remaining 3,517,300 for testing. With the Isolation Forest model, the average testing accuracy is 79.0% with average precision, recall and F-measure of 30.8%, 19.6% and 24.0% for negative logs, and 84.8%, 91.0% and 87.8% for positive logs, respectively. With Isolation Forest and one Autoencoder, the average testing accuracy is 82.9% with average precision, recall and F-measure of 49.9%, 49.6% and 49.3% for negative logs, and 89.8%, 90.7% and 89.9% for positive logs, respectively. With Isolation Forest and two Autoencoders, the average testing accuracy is 99.4% 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.

The precision, recall and F-measure results for negative logs are better than the 96%, 96% and 96%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. Experiments were also conducted with several well-known algorithms for anomaly detection. The average testing accuracy, precision, recall, F-measure and time for the Thunderbird data set with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM algorithms using 10-fold cross-validation are given in Table 2.1(c). The results show that the proposed model is significantly better than these algorithms.

4.2.4 Discussion

Isolation Forest has been shown to provide good results for anomaly detection problems [61]. However, it is used in this chapter to predict positive logs. Table 4.1(a) shows that Isolation Forest alone does not provide good results for the problem con-

sidered here, as the precision, recall and F-measure for the negative logs in all data sets are poor. However, the corresponding results for positive logs are much better. Table 4.1(b) shows the effect of feature extraction with an Autoencoder before Isolation Forest. These results indicate that most of the criteria for the positive and negative logs are improved for all three data sets. However, the negative log results are still poor. For the BGL data set, the average precision, recall and F-measure are improved from 28.9%, 36.6% and 32.3% to 42.7%, 76.5% and 54.3%, respectively, for negative logs while for positive logs the corresponding results are 94.8%, 92.8% and 93.8% to 98.0%, 91.9% and 94.9%. While the recall for the positive logs has decreased slightly, here precision is the most important criteria. For the Openstack data set, the average precision, recall and F-measure are mostly improved from 59.2%, 57.6% and 58.4% to 67.1%, 99.5% and 80.2% for negative logs and from 92.9%, 93.4% and 93.1% to 99.7%, 91.8% and 95.6% for positive logs while the recall for the positive logs has decreased. For the Thunderbird data set, the average precision, recall and F-measure are improved from 30.8%, 19.6% and 24.0% to 49.9%, 49.6% and 49.3%, respectively, for negative logs while for positive logs the corresponding results are 84.8%, 91.0% and 87.8% to 89.8%, 90.7% and 89.9%. Again, the recall for the positive logs has decreased slightly.

Precision for positive logs is the percentage of true positive logs detected out of all logs detected as positive, while recall for positive logs is the percentage of the positive logs that are detected. Table 4.1(b) shows that the precision for the positive logs is very reliable with 90% to 99% accuracy. Thus, it can be concluded that most logs which are predicted to be positive are correct. In other words, the number of negative logs predicted to be positive (F_P) is low with just Isolation Forest (especially with one Autoencoder). This reliable positive data (p_1) is used to train a second Autoencoder for anomaly detection with a threshold. Table 4.1(c) gives the test data results with

this Autoencoder which has been trained with p_1. Average precision, recall and F-measure for positive and negative logs are improved significantly for all data sets to more than 96%.

Isolation Forest with one Autoencoder is the most important step. Autoencoder networks are typically used for dimensionality reduction and so have fewer units in the hidden layers. However, a higher number of units in these layers is used so the input size is the same as the output. Binary cross-entropy is typically used in Autoencoder networks, but it was found that with categorical cross-entropy the averages of the feature values for each sample in o_2 are more separable into positive and negative logs, which is more useful in our case. This means that most of the positive logs have high average values and most of the negative logs have low average values. Further, it was noticed that the values for unimportant features (for example features that are repeated in the data set), are near zero with categorical cross-entropy and Relu activation without dimensionality reduction in the code layer. This indicates that the Autoencoder network works well for feature extraction with log messages.

The proposed unsupervised method has three advantages over supervised methods. First, knowledge via log message label is not required making this approach suitable for many practical applications. This is important as there are a vast variety of systems producing different log messages which makes it difficult to label data for supervised methods. Second, labeling data is a very time-consuming task because of the volumes of data and so is not a practical solution. Third, using an unsupervised method eliminates the human error inherent in labeling. Although supervised methods typically provide better performance than unsupervised methods, the proposed unsupervised method is better than the Improved K-Nearest Neighbors supervised algorithm [91] for BGL and Thunderbird data sets. This is because they use a simple supervised Machine Learning algorithm. Conversely, the proposed model employs

Deep Learning to extract features via deep Autoencoder networks.

Only a small amount of training data was used (less than 3% for BGL and Thunderbird, and 22% for Openstack for Isolation Forest with one Autoencoder, and less than 30% for BGL and Thunderbird for Isolation Forest with two Autoencoders), whereas training deep networks usually requires a significant amount of data for convergence. More training data was needed for the Openstack data set because it is small (around 25 times smaller than Thunderbird and 30 times smaller than BGL). The features extracted using the Autoencoder network were used as the input for Isolation Forest. This feature extraction required only a very small amount of training data, so the execution time was fast (1499 s for BGL, 1158 s for Thunderbird and 366 s for Openstack after pre-processing). Further, an Autoencoder network is a very fast Deep Learning algorithm, especially compared to LSTM networks, and the time complexity of Isolation Forest is linear [50]. The hyperparameters are not tuned so the results may be improved with tuning such as the learning rate and number of hidden layers.

4.3 Conclusion

Cloud systems are capable of generating millions of text log messages everyday. Thus, while the detection of anomalies in these logs is very important, the volume makes this a difficult task. In this chapter, a model was proposed for unsupervised anomaly detection using Isolation Forest and two deep Autoencoder networks. These networks are used for feature extraction and anomaly detection. Isolation Forest is often used for anomaly detection, but here it is used to predict positive data. The proposed model was evaluated using three well-known log message data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model is better

than other models employed in the literature. This is because training and feature extraction with the first Autoencoder network improves the Isolation Forest results, especially for positive log messages. Further, the number of negative logs predicted to be positive (F_p) is low with Isolation Forest, particularly with one Autoencoder. In the future, the performance of other models such as GMM and the effects of hyperparameter tuning can be investigated. Although an unsupervised model results were good, a detection threshold was required for each data set separately which was difficult to determine. To address this problem, a hybrid model which uses unsupervised models is proposed in the next chapter.

Table 4.1: The testing accuracy, precision, recall, F-measure and time for (a) Isolation Forest, (b) Isolation Forest with one Autoencoder and (c) Isolation Forest with two Autoencoders. The minimum, maximum and average (in parenthesis) values for the BGL, Openstack and Thunderbird data sets for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Data set	Testing Accuracy	Label	Precision	Recall	F-measure	Time (s)
(a)	**BGL**	88.6%-(88.7%)-88.8%	0	28.1%-(28.9%)-29.7%	35.2%-(36.6%)-38.1%	31.2%-(32.3%)-33.4%	276
			1	94.7%-(94.8%)-95.0%	92.8%-(92.8%)-92.9%	93.7%-(93.8%)-93.9%	
	Openstack	86.9%-(86.9%)-87.0%	0	54.5%-(59.2%)-64.0%	52.5%-(57.6%)-62.8%	53.5%-(58.4%)-63.4%	6
			1	92.1%-(92.9%)-93.8%	92.7%-(93.4%)-94.1%	92.4%-(93.1%)-93.9%	
	Thunderbird	78.8%-(79.0%)-79.3%	0	29.9%-(30.8%)-31.8%	19.3%-(19.6%)-20.0%	23.5%-(24.0%)-24.6%	158
			1	84.8%-(84.8%)-84.9%	90.8%-(91.0%)-91.3%	87.7%-(87.8%)-88.0%	
(b)	**BGL**	90.6%-(90.8%)-91.0%	0	40.2%-(42.7%)-45.2%	57.1%-(76.5%)-95.9%	47.2%-(54.3%)-61.4%	863
			1	96.5%-(98.0%)-99.6%	90.6%-(91.9%)-93.3%	94.9%-(94.9%)-94.9%	
	Openstack	92.9%-(92.9%)-93.0%	0	66.9%-(67.1%)-67.3%	99.4%-(99.5%)-99.7%	80.1%-(80.2%)-80.3%	171
			1	99.5%-(99.7%)-99.9%	91.8%-(91.8%)-91.9%	95.5%-(95.6%)-95.7%	
	Thunderbird	81.8%-(82.9%)-84.1%	0	45.6%-(49.9%)-54.2%	41.9%-(49.6%)-57.4%	43.7%-(49.3%)-54.9%	652
			1	88.4%-(89.8%)-91.2%	89.5%-(90.7%)-92.0%	89.1%-(89.9%)-90.8%	
(c)	**BGL**	99.4%-(**99.6%**)-99.9%	0	94.1%-(**96.8%**)-99.5%	97.8%-(**98.7%**)-99.6%	96.6%-(**98.1%**)-99.6%	1499
			1	99.8%-(99.8%)-99.9%	99.5%-(99.7%)-99.9%	99.7%-(99.8%)-99.9%	
	Openstack	98.4%-(**99.1%**)-99.7%	0	93.5%-(**96.1%**)-98.7%	95.7%-(**97.5%**)-99.3%	94.6%-(**96.8%**)-99.0%	366
			1	99.3%-(99.6%)-99.9%	98.9%-(99.3%)-99.8%	99.1%-(99.4%)-99.8%	
	Thunderbird	99.0%-(**99.4%**)-99.9%	0	94.7%-(**97.2%**)-99.8%	97.6%-(**98.6%**)-99.6%	97.2%-(**98.4%**)-99.7%	1158
			1	99.6%-(99.7%)-99.9%	98.9%-(99.4%)-99.9%	99.4%-(99.6%)-99.9%	

Chapter 5

Two Class Pruned Log Message Anomaly Detection

In this chapter, a hybrid model is proposed which uses the unsupervised K-means, GMM and BGM methods for data pruning and a supervised LSTM network for anomaly detection using reliable data. First, reliable positive logs are obtained using a GMM. Although GMMs are widely used for anomaly detection, a GMM is used here to prune only positive logs in the first step. Then reliable negative logs are obtained using pruning with the unsupervised K-means, GMM and BGM methods. Finally, a portion of the reliable positive and negative logs are used for anomaly detection using an LSTM network. The parameters of the proposed model are the same for all data sets. The proposed model is evaluated using the accuracy, precision, recall and F-measure criteria, and three log message data sets, namely BlueGene/L (BGL), Openstack and Thunderbird, are considered.

The steps of the proposed model are as follows.

1. An unsupervised algorithm is used with a GMM to select reliable positive logs.
2. An unsupervised algorithm which employs the K-means, GMM, and BGM methods iteratively is used to select reliable negative logs.

3. An LSTM network is used with the pruned logs for anomaly detection.

The rest of this chapter is organized as follows. In Section 5.1, the K-means, GMM, and BGM architectures are presented and the proposed model is described. Experimental results for the three data sets are given in Section 5.2 along with a discussion of the model performance. Finally, Section 5.3 provides some concluding remarks.

5.1 System Model

In this section, the K-means, GMM, and BGM architectures are given along with the proposed model.

5.1.1 K-means

K-means is an iterative clustering method. Given k classes, each cluster has a center which is the average of the samples in the cluster. The set of clusters is $S = \{S_1, S_2, \ldots, S_k\}$ and a sample is assigned to the cluster whose center it is closest to. First, the cluster centers are initialized randomly. Next, the Euclidean distances between each sample and the cluster centers, c_i, are calculated. Then, each sample is reassigned to the closest cluster and new centers are calculated for each cluster. This process continues until the clusters do not change or the maximum number of iterations is attained. This corresponds to minimizing the objective function given by

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - c_i||^2, \tag{5.1}$$

where k is the number of clusters, c_i is the center of cluster S_i, x is any data sample, and $||x - c_i||^2$ is the Euclidean distance from x to c_i.

5.1.2 Gaussian Mixture Model (GMM)

Gaussian Mixture Model (GMM) is a clustering method. It is assumed that each cluster consists of data with a normal (Gaussian) distribution. The goal is to estimate the distribution parameters of each cluster and determine the labels for the samples, i.e. which cluster each sample belongs to. The Expectation Maximization (EM) algorithm [11] can be used to obtain estimates for these parameters. The GMM Probability Density Function (PDF) of sample x_j is given by

$$p(x_j|\theta) = \sum_{i=1}^{m} \pi_i \mathcal{N}(x_j|\mu_i, \Sigma_i), \tag{5.2}$$

where π_i, μ_i and Σ_i are the weight, mean, and covariance matrices of the ith distribution, respectively, m is the number of distributions (clusters), and $\theta = \{\mu_i, \Sigma_i, \pi_i\}$ is the parameter set of the mixture model. For d features, the Gaussian distribution of x_j is

$$\mathcal{N}(x_j|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma_i|}} \exp\Big(-\frac{1}{2}(x_j - \mu_i)^T \Sigma_i^{-1}(x_j - \mu_i)\Big). \tag{5.3}$$

The EM algorithm iterates two steps, expectation (E-step) and maximization (M-step). First, the model parameters θ are randomly initialized and the expectation and maximization steps are repeated. The E-step is given by

$$\gamma_i^t(x_j) = \frac{\pi_i^t \mathcal{N}(x_j|\mu_i^t, \Sigma_i^t)}{\sum_{i=1}^{m} \pi_i^t \mathcal{N}(x_j|\mu_i^t, \Sigma_i^t)}, \tag{5.4}$$

where t is the iteration number. Then the parameters are estimated in the M-step as

$$\mu_i^{t+1} = \frac{\sum_{j=1}^{N} \gamma_i^t(x_j) x_j}{\sum_{j=1}^{N} \gamma_i^t(x_j)}, \tag{5.5}$$

$$\Sigma_i^{t+1} = \frac{\sum_{j=1}^{N} \gamma_i^t(x_j)(x_j - \mu_i^{t+1})(x_j - \mu_i^{t+1})^T}{\sum_{j=1}^{N} \gamma_i^t(x_j)}, \tag{5.6}$$

$$\pi_i^{t+1} = \frac{1}{N} \sum_{j=1}^{N} \gamma_i^t(x_j). \tag{5.7}$$

where N is the number of samples. These steps are repeated until the criteria are satisfied or a maximum number of iterations is reached.

5.1.3 Dirichlet Process Gaussian Mixture Model (BGM)

The Dirichlet Process Gaussian Mixture Model (BGM) is a non-parametric Bayesian mixture model that is an extension of finite mixture models. The number of clusters (classes) does not need to be explicitly predefined because it is a non-parametric model. BGM uses the Dirichlet Process (DP) which is a generalized form of a Dirichlet distribution [16]. A DP is composed of a base distribution G_0 and a positive concentration scaler α. Because this model is not a finite mixture model, variational inference is employed [7]. The model parameters are

$$\pi_i \sim DP(G_0, \alpha), \tag{5.8}$$

$$\Sigma_i \sim \mathcal{W}^{-1}(v, s), \tag{5.9}$$

$$\mu_i \sim \mathcal{N}(\mu_0, \Sigma_i), \tag{5.10}$$

where μ_0 and Σ_i are the mean and covariance of the Gaussian distribution, s is the scale matrix, and v is the number of degrees of freedom for the Inverse-Wishart distribution [93].

5.1.4 Proposed Model

The proposed model architecture has three steps. First, positive logs are selected using an unsupervised GMM. Second, negative logs are selected through multiple rounds of the unsupervised GMM, BGM, and K-means methods. Finally, anomalies are detected using an LSTM network with the selected (reliable) positive and negative logs.

First, simple text pre-processing including changing letters to lowercase, removing hyphens and tokenization are applied to the data set D. Next, the sentences are padded to 40 tokens, and sentences including less than five tokens are removed. Then, the number of appearances of each token in the data set is computed and the tokens are ordered from most frequent to least frequent. Each token is given an index starting from zero and the indices are used to replace the tokens in the data set. Next, the data set is normalized so all values are between 0 and 1 and the entries are shuffled. Then D is divided into two sets, t_1 with 2% of the data for training and r_1 with the remaining 98% of the data. The set t_1 is small to keep the computational complexity low and have more data for the rest of the algorithm. The proportion of negative and positive logs in these sets is the same as in D.

Select Reliable Positive Logs

A GMM is used to prune the positive logs. It is trained with t_1 and tested with r_1. The negative predicted logs (predicted output $y = 0$) and positive predicted logs (predicted output $y = 1$) are counted and labeled c_0 and c_1, respectively. If the number of logs predicted as positive is less than the number predicted as negative, then c_0 and c_1 are swapped. This is because it is known that the number of anomalies (negative logs) is

much less than the number of positive logs (around 10%). The variance is given by

$$var = \frac{\sum_{i=1}^{F}(x_i - \hat{x})^2}{F} \tag{5.11}$$

where x_i is the ith feature, $\hat{x}$ is the average of the features and F is the total number of features in the data set. Let $a = \frac{c_1}{c_0}$ and the variances of the negative and positive predicted logs be z_{var} and o_{var}, respectively. If $a > 3$ and $o_{var} \times c < z_{var}$ (c is a constant), then the positive and negative predicted logs are added to the sets o_0 (reliable positive logs) and z_0 (rest of the data), respectively. The threshold for a was chosen considering that the majority of the logs are positive. A high value of c increases the probability of getting only positive logs but if it is too high the algorithm criterion may not be satisfied. It was set to $c = 1.6$ for all data sets based on the experimental results obtained.

The variance measures the spread of a data set. If the model predicted most of the positive logs correctly (small number of false positives), then the variance of the positive logs should be lower than that of the negative logs. A high variance may indicate that there is a mix of positive and negative logs whereas a small variance indicates that there are mostly positive logs predicted correctly. If the conditions are met, the results are kept, otherwise the process is repeated.

Select Reliable Negative Logs

The GMM, K-means and BGM methods are now used to select negative logs. These models were chosen because they are efficient unsupervised models. There are n rounds and in each round, GMM, K-means and BGM are run m times. In the first round, the models are trained with z_0 from the previous step. In subsequent rounds, the results from the previous round z are used for this purpose. The entropy of sample

x_j is given by

$$H = -\sum_{i=1}^{d} \frac{n_i}{M} \ln(\frac{n_i}{M}), \tag{5.12}$$

where n_i is the ith feature of the sample, d is the length of the sample and M is the sum of the sample features.

For a model run, denote the average entropy of the logs predicted as negative and positive as sh_0 and sh_1, respectively. If $sh_0 < sh_1$ then the predicted negative logs are appended to z_1, and the predicted positive logs are appended to o_1. This is because small features appear frequently in positive logs so they are more uniform and thus have a higher entropy. At the end of a round, z_1 is assigned to z for use in the next round. The logs in z are counted and ordered from most frequent to least frequent and the repetitions discarded. This is done so that each log appears at most once in z and the logs that appear more often are used earlier so the models in the next round can predict better. The logs in z_1 are discarded at the start of each round.

In each round, the prediction of positive and negative logs is done using z. Thus, the number of positive logs is reduced and only reliable negative logs are kept. The final z_1 contains the predicted negative logs from the last round and these are used in the next step.

Anomaly Prediction

In this step, an LSTM network is used with the reliable negative (z_1) and positive (o_0) logs from the previous two steps for anomaly detection. Each log in z_1 and o_1 is counted and ordered from most frequent to least frequent and the repetitions are discarded. The first L logs of z_1 are selected and assigned to z_2 (most reliable predicted negative logs), and the remaining logs are assigned to o_2. Logs which appear in o_1 but not in z_1 are placed in o_3. The reliable positive logs o_0 obtained in the first step are shuffled, and 10% are randomly assigned to o_4 and the remainder to o_5. The

logs in z_2 are repeated four times and assigned to x_n. A portion of o_4 which is the same size as the number of elements in x_n is randomly chosen and assigned to x_p. Thus, the reliable negative logs are oversampled so there are the same number of reliable positive and negative logs. This is because LSTM networks work better with balanced data and should be trained with a sufficient number of positive and negative logs. The logs in x_n and x_p are labeled with $y = 0$ and $y = 1$ indicating negative and positive logs, respectively, and x_n and x_p are assigned to t_2. The remaining logs in o_4 are assigned to o_6, and o_2, o_3, o_5 and o_6 are assigned to t_3. The data set was initially scaled so all values are between 0 and 1, but this is reversed for t_2 and t_3 to provide training and testing sets, respectively, for the LSTM network.

The LSTM network is trained with 90% of t_2, validated with the remaining 10% of t_2, and tested with t_3. The parameters used are $k = 20$ and $n = m = 5$, and $L = 10000$ for the BGL and Thunderbird data sets and $L = 3000$ for the Openstack data set. A different value of L is used because the Openstack data set is smaller than the BGL and Thunderbird data sets. For training the BGL and Thunderbird data sets, an LSTM network with three hidden layers of size 256, batch size 128 and a maximum of 10 training epochs is used. To prevent overfitting, Dropout with probability 0.5 and early stopping are used. The softmax activation function is applied in the last dense layer. The cross-entropy loss function and Adam optimizer are used for training. The Adam optimizer is used because it has been shown to provide good performance and fast convergence in DL algorithms [81]. For the Openstack data set, an LSTM network with a single hidden layer of size 512 and embedding dimension of size 512 is used. A single layer network is used for this data set because it is smaller than the other data sets. The rest of the architecture for the Openstack data set is the same as above. All network parameters were chosen based on the experimental results obtained. An LSTM network is used for anomaly detection because it has

been shown to provide good results in classifying sequential data [23]. However, other DL discriminative networks such as a Convolutional Neural Network (CNN) can be employed. The proposed model algorithms are given in Algorithms 3-5. The data preparation for Algorithm 5 is shown in Fig. 5.1.

Algorithm 3 Prune Positive Logs

Input: Data set D scaled between $[0, 1]$, training set $t_1 = 2\%$ of D, remaining set $r_1 = 98\%$ of D

for $i \leftarrow 1$ **to** k **do**
    Train GMM with t_1 and test with r_1
    Count negative predicted logs $(y = 0)$ (c_0)
    Count positive predicted logs $(y = 1)$ (c_1)
    if $c_1 < c_0$ **then** swap c_0 and c_1
    $a = \frac{c_1}{c_0}$
    Compute variance of negative predicted logs (z_{var})
    Compute variance of positive predicted logs (o_{var})
    if $(a > 3)$ **and** $(o_{var} \times 1.6 < z_{var})$ **then**
        $z_0 \leftarrow$ Selected negative predicted logs
        $o_0 \leftarrow$ Selected positive predicted logs
        break for loop

Algorithm 4 Prune Negative Logs

Model ← [GMM, K-means, BGM]
for $round \leftarrow 1$ **to** n **do**
    $z_1 \leftarrow Empty$
    for each Model **do**
        for $j \leftarrow 1$ **to** m **do**
            if $round = 1$ **then**
                Train on z_0. Get negative and positive predicted logs
            else
                Train on z. Get negative and positive predicted logs
            Compute average entropy of negative predicted logs (sh_0)
            Compute average entropy of positive predicted logs (sh_1)
            if $sh_0 < sh_1$ **then**
                Append negative predicted logs to z_1
                Append positive predicted logs to o_1
    $z = z_1$
    Count each log in z, then order from most frequent to least frequent and delete the repetitions

Algorithm 5 Anomaly Prediction

1: Count each log in o_1, then order from most frequent to least frequent and delete the repetitions

2: Count each log in z_1, then order from most frequent to least frequent and delete the repetitions

3: $z_2 \leftarrow$ first L logs of z_1

4: $o_2 \leftarrow$ rest of z_1

5: $o_3 \leftarrow o_1$ minus any logs that also appear in z_1

6: $o_4 \leftarrow 10\%$ of o_0 chosen randomly

7: $o_5 \leftarrow$ rest of o_0

8: $x_n \leftarrow z_2$ repeated four times, label as $y = 0$ (oversampling)

9: $x_p \leftarrow$ a random portion of o_4 so the length is the same as x_n, label as $y = 1$

10: $t_2 \leftarrow x_n$ and x_p

11: $o_6 \leftarrow$ rest of o_4

12: $t_3 \leftarrow o_2$, o_3, o_5, and o_6

13: Reverse the scaling of t_2 and t_3

14: Train the LSTM network using t_2 and test using t_3

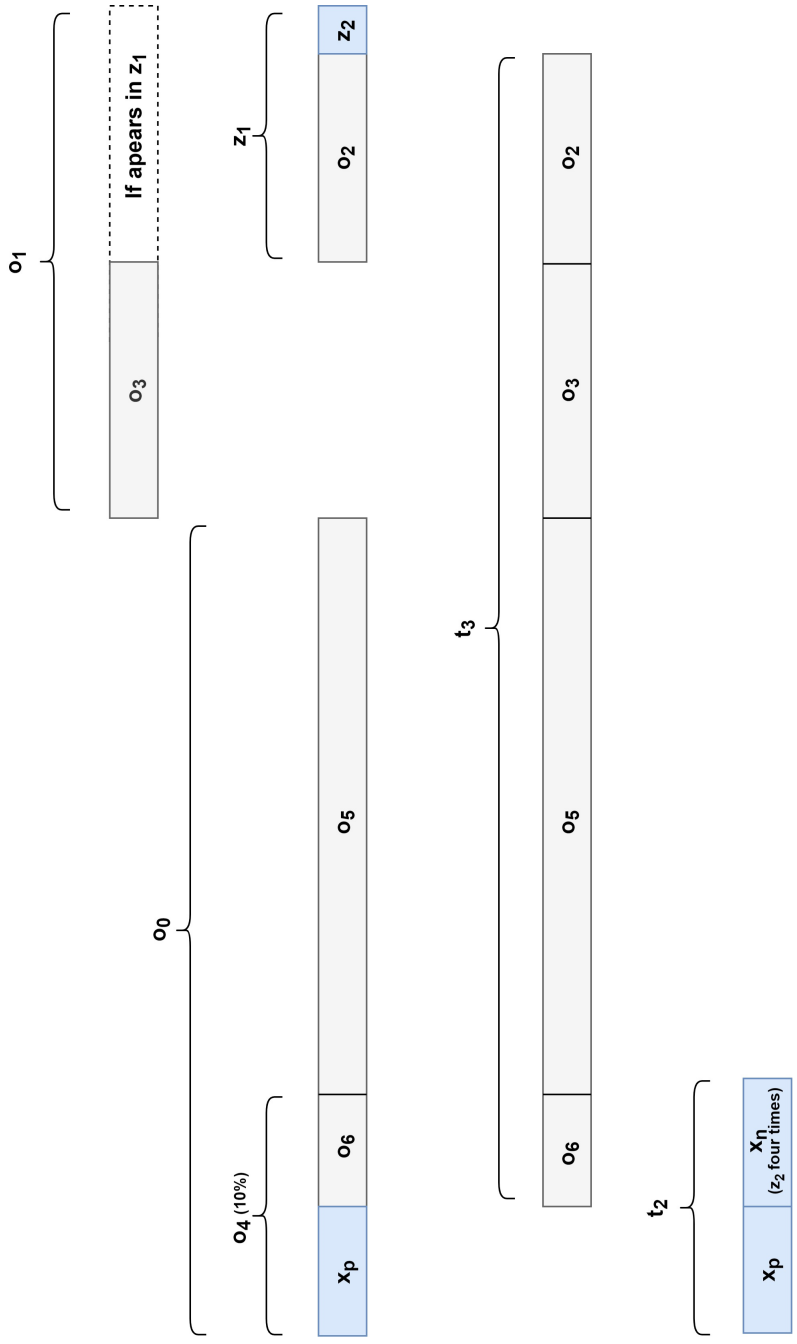

Figure 5.1: The data preparation for Algorithm 5.

5.2 Results

In this section, the proposed model is evaluated using the BGL, Openstack and Thunderbird data sets. Four performance criteria are considered, namely accuracy, precision, recall and F-measure. All experiments were run on the Compute Canada Graham cluster with 32 CPU cores, two P100 GPUs and 124 GB of memory. The algorithms were implemented using Python, Keras and Scikit-learn.

The hyperparameters of the proposed model were not tuned so the default values were used in all experiments. Each experiment was repeated 10 times and the minimum, maximum and average testing accuracy, precision, recall, F-measure and time were obtained. Table 5.1(a) gives the proposed model results for the BGL, Openstack and Thunderbird data sets using GMM for positive pruning. For comparison, the proposed model results using BGM for positive pruning are given in Table 5.1(b) with the order for negative pruning changed to K-means, GMM, and BGM. Table 5.2 presents the positive logs pruning results for the BGL, Openstack and Thunderbird data sets with (a) GMM and (b) BGM. Tables 5.3 to 5.5 give the negative logs pruning results for the BGL, Openstack and Thunderbird data sets, respectively, with (a) GMM, (b) K-means, and (c) BGM for $n = 5$ rounds.

5.2.1 BGL

The BlueGene/L (BGL) data set has 4,399,502 positive logs and 348,460 negative logs. From these, 94,960 logs are used for the training set t_1 and 4,653,002 for the remaining set r_1. Using GMM for positive pruning, the final average testing accuracy is 99.5% 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. Using BGM for positive pruning, the final average testing accuracy is 99.5% with average

precision, recall and F-measure of 95.4%, 98.5% and 96.9% for negative logs, and 99.9%, 99.6% and 99.7% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbors (IKNN) supervised algorithm [91]. The precision, recall and F-measure results for negative log messages are also better than the 83%, 99% and 91%, respectively, for SVM unsupervised learning [30], and the 82.5%, 94.7% and 88.2%, respectively, for the nLSALog algorithm [98]. Several well-known models were also evaluated for anomaly detection. The average testing accuracy, precision, recall, F-measure and time with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM methods for the BGL data set using 10-fold cross-validation are given in Table 2.1(a). These results show that the proposed model is much better than these models.

5.2.2 Openstack

The Openstack data set has 137,074 positive log messages and 18,434 negative log messages. From these, 3,111 logs are used for the training set t_1 and 152,397 for the remaining set r_1. Using GMM for positive pruning, the final average testing accuracy is 99.9% with average precision, recall and F-measure of 99.9%, 99.7% and 99.8% for negative logs, and 99.9%, 99.9% and 99.9% for positive logs, respectively. Using BGM for positive pruning, the final average testing accuracy is 99.9% with average precision, recall and F-measure of 99.7%, 99.9% and 99.8% for negative logs, and 99.9%, 99.9% and 99.9% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 94%, 99% and 97% obtained with the Deeplog network [14]. Several well-known models were also evaluated for anomaly detection. The average testing accuracy, precision, recall, F-measure and time with the BGM, EEnvelope, GMM, K-means,

LOF, and OC-SVM methods for the Openstack data set using 10-fold cross-validation are given in Table 2.1(b). This shows that the results for the proposed model are much better than those for the other models.

5.2.3 Thunderbird

From the Thunderbird data set, 3,000,000 positive log messages and 324,824 negative log messages are used. Of these, 66,497 messages are used for the training set t_1 and 3,258,327 for the remaining set r_1. Using GMM for positive pruning, the final average testing accuracy is 99.8% with average precision, recall and F-measure of 98.9%, 99.6% and 99.3% for negative logs, and 99.9%, 99.8% and 99.9% for positive logs, respectively. Using BGM for positive pruning, the final average testing accuracy is 99.8% with average precision, recall and F-measure of 99%, 99.6% and 99.3% for negative logs, and 99.9%, 99.8% and 99.9% for positive logs, respectively.

The precision, recall and F-measure results for negative logs are better than the 96% for all criteria with the IKNN supervised algorithm [91]. Several well-known models were also evaluated for anomaly detection. The average testing accuracy, precision, recall, F-measure and time with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM methods for the Thunderbird data set using 10-fold cross-validation are given in Table 2.1(c). This shows that the results for the proposed model are much better than the results for these models.

5.2.4 Discussion

GMM, BGM, and K-means are well-known clustering algorithms. Clustering algorithms have been shown to provide good results with text data [1]. However, a GMM is used here for pruning positive logs and the unsupervised GMM, BGM, and K-means methods are used for pruning negative logs. This eliminates the need to label

log messages to detect anomalies. The amount of positive data is far greater than the amount of negative data, so the positive data can be accurately predicted using clustering algorithms. However, the negative cluster contains a lot of positive data and this is a disadvantage of using clustering methods with imbalanced data [43]. We take advantage of this data imbalance as GMM can easily detect the positive data. Another disadvantage of unsupervised clustering is that clusters may be labeled incorrectly [59]. Thus, not only do negative clusters include positive logs but clusters may be incorrectly labeled in different runs. As a consequence, the average results shown in Table 2.1 for the BGM, GMM and K-means methods are poor. Comparing the results in Tables 5.1 and 2.1, it is evident that BGM, GMM and K-means alone do not provide good log message anomaly detection results. This is also due to the complexity of the unstructured log messages. For negative pruning, our experimental results indicate that even though the models are randomly initialized, using just one model can limit the pruning process so that many unreliable logs are retained. Thus, multiple models are employed.

The precision for positive logs is the percentage of true positive logs predicted of all logs predicted to be positive. Table 5.2 shows that with GMM and BGM for positive pruning, the average precision of positive logs is 99.9% for the BGL, Openstack and Thunderbird data sets, which is very high. Thus, most logs that are predicted to be positive are correct, and the number of negative log messages predicted to be positive (F_P) is low. This indicates that pruning positive logs using Algorithm 3 is effective. However, the average precision for negative logs for the BGL and Thunderbird data sets is around 76% and 50%, respectively, which is quite low.

In separate experiments, the value of the constant c and the threshold for a were varied to determine their effect on Algorithm 3. The criterion for this algorithm is that the set z_0 is not empty. This is because Algorithm 4 requires this data for

training. For the Openstack and Thunderbird data sets, this criterion was satisfied for $c = 1.4, 1.5, 1.6, 1.7$ and 1.8 with $a > 3$. For the BGL data set, it was satisfied for all values except $c = 1.8$. For the BGL and Openstack data sets, the criterion was satisfied for $a > 3, 4$, and 6 with $c = 1.6$, but not for $a > 10$. For the Thunderbird data set, it was not satisfied for $a > 6$ and 10.

The effect of the negative logs pruning algorithm is shown in Tables 5.3 to 5.5 for the BGL, Openstack and Thunderbird data sets (with a GMM for positive pruning), respectively. Here, precision for the negative logs is the most important criterion. For the BGL data set, the average precision of the negative logs with the GMM, K-means and BGM methods increased from 82.4% to 100%, 80.8% to 95.9% and 88.1% to 100%, respectively, over the five rounds. For the Openstack data set, the average precision of the negative logs with the GMM, K-means and BGM methods increased from 99.9% to 100% for all models over the five rounds. For the Thunderbird data set, the average precision of the negative logs with the GMM, K-means and BGM methods was approximately the same, 99.9% to 100%, over the five rounds. These results indicate that Algorithm 4 is very effective in pruning negative logs. Further, the BGL data set required five rounds to obtain good results but only two rounds were sufficient for the Openstack and Thunderbird data sets. The GMM, K-means and BGM methods were used here to prune negative logs, but other unsupervised models can be added.

The anomaly detection results with the LSTM network using the reliable positive and negative logs are shown in Table 5.1. The final results with GMM and BGM positive log pruning were similar. The proposed hybrid model (with unsupervised selection of reliable logs), has three advantages over supervised methods. First, it is suitable for many practical applications as there is no need to label data. Second, labeling data is a time-consuming task and in many cases is not feasible. Third,

using an unsupervised method eliminates the human error inherent in labeling. The default hyperparameters were used with the proposed model so better results may be obtained with hyperparameter tuning.

5.3 Conclusion

Many millions of log messages are generated each day in cloud and other systems. These messages are important for system maintenance which includes anomaly detection. Log messages consist of unstructured data which is mostly text. Thus, ML is a good choice for anomaly detection. In this chapter, a hybrid log message anomaly detection technique using DL was proposed with pruning of positive and negative log messages. An unsupervised algorithm with a GMM was used to prune positive logs. Then, an unsupervised algorithm was used to prune negative logs using the K-means, GMM, and BGM methods iteratively. The precision with the pruning algorithms for positive and negative logs was high, i.e. there were few false positives (F_P). The proposed model was tested on three different log message data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model is better than other well-known approaches. Future research can consider the effect of adding other unsupervised methods such as Isolation Forest to the proposed model. Further, a CNN network can be used for anomaly detection instead of an LSTM network and hyperparameter tuning can be investigated. In the next chapter, a very small amount of data is used with FCM and MLP models to select reliable logs to obtain a fast algorithm.

Table 5.1: The proposed model testing accuracy, precision, recall, F-measure, and average time with (a) GMM for positive log pruning and (b) BGM for positive log pruning. The minimum, maximum and average (in parenthesis) values are given for 10 runs with the BGL, Openstack and Thunderbird data sets. Positive labels are denoted by 1 and negative labels by 0.

	Data set	Testing Accuracy	Label	Precision	Recall	F-measure	Time (s)
(a)	**BGL**	99.3%-(**99.5%**)-99.6%	0	93.2%-(**95.6%**)-97.7%	97.1%-(**97.8%**)-98.7%	95.6%-(**96.7%**)-97.6%	3725
			1	99.8%-(99.8%)-99.9%	99.4%-(99.6%)-99.8%	99.6%-(99.7%)-99.8%	
	Openstack	99.8%-(**99.9%**)-100%	0	99.3%-(**99.9%**)-100%	97.9%-(**99.7%**)-100%	99.0%-(**99.8%**)-100%	177
			1	99.7%-(99.9%)-100%	99.9%-(99.9%)-100%	99.9%-(99.9%)-100%	
	Thunderbird	99.6%-(**99.8%**)-99.9%	0	97.1%-(**98.9%**)-99.9%	99.6%-(**99.6%**)-99.7%	98.3%-(**99.3%**)-99.8%	3550
			1	99.9%-(99.9%)-99.9%	99.5%-(99.8%)-99.9%	99.7%-(99.9%)-99.9%	
(b)	**BGL**	99.4%-(**99.5%**)-99.7%	0	94.0%-(**95.4%**)-97.4%	97.2%-(**98.5%**)-99.4%	96.0%-(**96.9%**)-98.2%	3649
			1	99.8%-(99.9%)-99.9%	99.5%-(99.6%)-99.8%	99.7%-(99.7%)-99.8%	
	Openstack	99.6%-(**99.9%**)-100%	0	96.7%-(**99.7%**)-100%	99.6%-(**99.9%**)-100%	98.3%-(**99.8%**)-100%	134
			1	99.9%-(99.9%)-100%	99.6%-(99.9%)-100%	99.8%-(99.9%)-100%	
	Thunderbird	99.7%-(**99.8%**)-99.9%	0	97.5%-(**99.0%**)-99.9%	99.6%-(**99.6%**)-99.7%	98.6%-(**99.3%**)-99.8%	3136
			1	99.9%-(99.9%)-99.9%	99.7%-(99.8%)-99.9%	99.8%-(99.9%)-99.9%	

Table 5.2: Positive pruning testing accuracy, precision, recall, and F-measure with (a) GMM and (b) BGM for the BGL, Openstack and Thunderbird data sets. The minimum, maximum, and average (in parenthesis) values are given for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Data set	Testing Accuracy	Label	Precision	Recall	F-measure
(a)	**BGL**	97.1%-(97.8%)-98.4%	0	71.6%-(76.8%)-81.9%	99.9%-(99.9%)-100%	83.4%-(86.9%)-90.0%
			1	99.9%-(**99.9%**)-100%	96.8%-(97.6%)-98.2%	98.4%-(98.8%)-99.1%
	Openstack	99.8%-(99.9%)-100%	0	98.8%-(99.8%)-100%	99.3%-(99.8%)-100%	99.1%-(99.8%)-100%
			1	99.9%-(**99.9%**)-100%	99.8%-(99.9%)-100%	99.9%-(99.9%)-100%
	Thunderbird	89.9%-(90.0%)-90.5%	0	49.3%-(50.4%)-59.6%	99.9%-(99.9%)-100%	66.0%-(67.0%)-74.7%
			1	99.9%-(**99.9%**)-100%	88.8%-(88.9%)-89.0%	94.1%-(94.1%)-94.2%
(b)	**BGL**	97.3%-(97.6%)-98.2%	0	73.1%-(75.3%)-80.6%	99.9%-(99.9%)-100%	84.5%-(85.9%)-89.2%
			1	99.9%-(**99.9%**)-100%	97.1%-(97.4%)-98.1%	98.5%-(98.7%)-99.0%
	Openstack	99.8%-(99.9%)-99.9%	0	98.5%-(99.4%)-99.9%	99.6%-(99.8%)-99.9%	99.1%-(99.6%)-99.9%
			1	99.9%-(**99.9%**)-99.9%	99.8%-(99.9%)-99.9%	99.9%-(99.9%)-99.9%
	Thunderbird	89.9%-(90.0%)-90.1%	0	49.3%-(49.5%)-49.8%	99.9%-(99.9%)-100%	66.0%-(66.2%)-66.5%
			1	99.9%-(**99.9%**)-100%	88.9%-(88.9%)-89.1%	94.1%-(94.1%)-94.2%

Table 5.3: Negative pruning testing accuracy, precision, recall, and F-measure for the BGL data set with (a) GMM, (b) K-means and (c) BGM methods for five rounds (with a GMM for positive pruning). The minimum, maximum and average (in parenthesis) values are given for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Round	Testing Accuracy	Label	Precision	Recall	F-measure
(a)	**1**	76.2%-(82.6%)-88.9%	0	75.0%-(82.4%)-100%	81.5%-(99.0%)-100%	85.7%-(89.7%)-93.1%
			1	55.8%-(97.4%)-100%	16.1%-(29.1%)-100%	27.8%-(40.9%)-71.9%
	2	85.4%-(96.0%)-99.2%	0	99.9%-(99.9%)-100%	81.5%-(94.9%)-99.0%	89.8%-(97.3%)-99.5%
			1	58.6%-(86.2%)-97.1%	99.9%-(99.9%)-100%	73.9%-(91.8%)-98.5%
	3	81.9%-(90.6%)-99.5%	0	99.3%-(99.9%)-100%	81.0%-(90.2%)-99.5%	89.5%-(94.6%)-99.7%
			1	19.7%-(47.3%)-89.8%	86.6%-(99.3%)-100%	32.9%-(59.1%)-94.6%
	4	81.9%-(91.1%)-99.5%	0	99.9%-(100%)-100%	81.0%-(90.7%)-99.5%	89.5%-(94.9%)-99.7%
			1	19.4%-(50.9%)-90.1%	99.9%-(99.9%)-100%	32.5%-(62.1%)-94.8%
	5	81.9%-(88.7%)-99.0%	0	100%-(**100%**)-100%	81.0%-(88.2%)-98.9%	89.5%-(93.5%)-99.5%
			1	19.6%-(43.7%)-82.3%	100%-(100%)-100%	32.7%-(55.1%)-90.3%
(b)	**1**	76.1%-(81.8%)-87.1%	0	75.0%-(80.8%)-86.4%	99.9%-(99.9%)-100%	85.7%-(89.4%)-92.7%
			1	99.9%-(99.9%)-100%	16.0%-(21.7%)-28.9%	27.6%-(35.5%)-44.8%
	2	96.0%-(96.2%)-96.3%	0	95.3%-(95.5%)-95.7%	99.9%-(99.9%)-100%	97.6%-(97.7%)-97.8%
			1	99.9%-(99.9%)-100%	72.6%-(80.8%)-85.3%	84.1%-(89.3%)-92.1%
	3	86.9%-(87.7%)-91.1%	0	94.9%-(95.8%)-99.9%	91.1%-(91.2%)-91.2%	93.0%-(93.4%)-95.3%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	86.9%-(87.5%)-90.8%	0	94.9%-(95.6%)-99.9%	90.8%-(91.1%)-91.2%	93.0%-(93.3%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	87.0%-(87.7%)-90.8%	0	95.0%-(**95.9%**)-99.9%	90.8%-(91.1%)-91.2%	93.0%-(93.4%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
(c)	**1**	76.2%-(88.2%)-96.1%	0	75.0%-(88.1%)-100%	81.5%-(98.4%)-100%	85.7%-(92.7%)-97.6%
			1	55.8%-(94.3%)-100%	16.1%-(57.5%)-100%	27.8%-(67.5%)-89.7%
	2	85.4%-(96.0%)-99.2%	0	99.9%-(99.9%)-100%	81.5%-(94.9%)-99.0%	89.8%-(97.3%)-99.5%
			1	58.6%-(86.2%)-97.1%	99.9%-(99.9%)-100%	73.9%-(91.8%)-98.5%
	3	81.9%-(90.6%)-99.5%	0	99.3%-(99.9%)-100%	81.0%-(90.2%)-99.5%	89.5%-(94.6%)-99.7%
			1	19.7%-(47.3%)-89.8%	86.6%-(99.3%)-100%	32.9%-(59.1%)-94.6%
	4	67.3%-(89.8%)-99.5%	0	99.3%-(99.9%)-100%	67.3%-(89.4%)-99.5%	80.4%-(94.1%)-99.7%
			1	0.1%-(47.2%)-90.1%	86.6%-(99.3%)-100%	0.1%-(57.9%)-94.8%
	5	81.9%-(88.7%)-99.0%	0	100%-(**100%**)-100%	81.0%-(88.2%)-98.9%	89.5%-(93.5%)-99.5%
			1	19.6%-(43.7%)-82.3%	100%-(100%)-100%	32.7%-(55.1%)-90.3%

Table 5.4: Negative pruning testing accuracy, precision, recall, and F-measure for the Openstack data set with (a) GMM, (b) K-means and (c) BGM methods for five rounds (with a GMM for positive pruning). The minimum, maximum and average (in parenthesis) values are given for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Round	Testing Accuracy	Label	Precision	Recall	F-measure
(a)	**1**	61.5%-(64.3%)-94.0%	0	99.1%-(99.9%)-100%	61.5%-(64.3%)-94.0%	76.2%-(78.0%)-96.9%
			1	0.0%-(0.1%)-1.6%	0.0%-(4.2%)-52.8%	0.0%-(0.2%)-3.1%
	2	61.5%-(83.9%)-94.0%	0	100%-(100%)-100%	61.5%-(83.8%)-94.0%	76.2%-(90.7%)-96.9%
			1	0.0%-(0.7%)-9.3%	0.0%-(7.5%)-100%	0.0%-(1.3%)-17.1%
	3	64.9%-(88.8%)-98.3%	0	100%-(100%)-100%	64.9%-(88.8%)-98.3%	78.7%-(93.8%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	71.6%-(92.0%)-98.2%	0	100%-(100%)-100%	71.6%-(92.0%)-98.2%	83.4%-(95.7%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	58.7%-(88.6%)-98.2%	0	100%-(**100%**)-100%	58.7%-(88.6%)-98.2%	74.0%-(93.6%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
(b)	**1**	62.7%-(62.8%)-62.9%	0	99.0%-(99.9%)-100%	62.7%-(62.8%)-62.9%	76.9%-(77.1%)-77.3%
			1	0.0%-(0.1%)-1.4%	0.0%-(4.5%)-46.7%	0.0%-(0.3%)-2.8%
	2	62.8%-(84.1%)-90.4%	0	100%-(100%)-100%	62.8%-(84.1%)-90.4%	77.1%-(90.9%)-95.0%
			1	0.0%-(1.0%)-9.3%	0.0%-(11.4%)-100%	0.0%-(1.9%)-17.1%
	3	90.4%-(91.6%)-94.6%	0	100%-(100%)-100%	90.4%-(91.6%)-94.6%	95.0%-(95.6%)-97.2%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	91.4%-(93.2%)-94.6%	0	100%-(100%)-100%	91.4%-(93.2%)-94.6%	95.5%-(96.5%)-97.2%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	78.6%-(91.8%)-94.6%	0	100%-(**100%**)-100%	78.6%-(91.8%)-94.6%	88.0%-(95.7%)-97.2%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
(c)	**1**	61.5%-(64.3%)-94.0%	0	99.1%-(99.9%)-100%	61.5%-(64.3%)-94.0%	76.1%-(78.0%)-96.9%
			1	0.0%-(0.1%)-1.6%	0.0%-(4.3%)-53.8%	0.0%-(0.2%)-3.2%
	2	61.5%-(82.0%)-90.4%	0	100%-(100%)-100%	61.5%-(82.0%)-90.4%	76.1%-(89.6%)-95.0%
			1	0.0%-(0.8%)-9.3%	0.0%-(8.8%)-100%	0.0%-(1.5%)-17.1%
	3	53.6%-(83.3%)-98.3%	0	100%-(100%)-100%	53.6%-(83.3%)-98.3%	69.8%-(90.1%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	71.6%-(89.8%)-98.2%	0	100%-(100%)-100%	71.6%-(89.8%)-98.2%	83.4%-(94.5%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	54.4%-(84.8%)-98.2%	0	100%-(**100%**)-100%	54.4%-(84.8%)-98.2%	70.5%-(91.1%)-99.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%

Table 5.5: Negative pruning testing accuracy, precision, recall, and F-measure for the Thunderbird data set with (a) GMM, (b) K-means and (c) BGM methods for five rounds (with a GMM for positive pruning). The minimum, maximum and average (in parenthesis) values are given for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Round	Testing Accuracy	Label	Precision	Recall	F-measure
(a)	**1**	99.7%-(99.8%)-99.8%	0	99.9%-(99.9%)-100%	99.6%-(99.6%)-99.6%	99.8%-(99.8%)-99.8%
			1	99.4%-(99.6%)-99.6%	99.9%-(99.9%)-100%	99.7%-(99.8%)-99.8%
	2	84.6%-(88.1%)-88.4%	0	99.9%-(99.9%)-100%	84.6%-(88.1%)-88.4%	91.7%-(93.7%)-93.9%
			1	0.0%-(0.1%)-0.1%	0.0%-(10.0%)-100%	0.0%-(0.1%)-0.1%
	3	90.0%-(90.6%)-90.7%	0	99.9%-(99.9%)-100%	90.0%-(90.6%)-90.7%	94.7%-(95.1%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	45.2%-(82.2%)-90.7%	0	99.9%-(99.9%)-100%	45.2%-(82.2%)-90.7%	62.3%-(89.0%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	45.2%-(83.0%)-90.7%	0	99.9%-(**99.9%**)-100%	45.2%-(83.0%)-90.7%	62.3%-(89.9%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
(b)	**1**	70.9%-(78.0%)-99.8%	0	100%-(100%)-100%	51.2%-(57.0%)-99.6%	67.7%-(72.1%)-99.8%
			1	58.1%-(69.4%)-99.6%	100%-(100%)-100%	73.5%-(81.7%)-99.8%
	2	35.0%-(35.3%)-35.8%	0	100%-(100%)-100%	35.0%-(35.3%)-35.8%	51.8%-(52.2%)-52.7%
			1	0.0%-(0.1%)-0.1%	0.0%-(17.4%)-100%	0.0%-(0.1%)-0.1%
	3	36.1%-(36.5%)-37.9%	0	100%-(100%)-100%	36.1%-(36.5%)-37.9%	53.0%-(53.4%)-55.0%
			1	0.0%-(0.1%)-0.1%	0.0%-(5.9%)-100%	0.0%-(0.1%)-0.1%
	4	36.1%-(36.6%)-38.0%	0	100%-(100%)-100%	36.1%-(36.6%)-38.0%	53.0%-(53.6%)-55.0%
			1	0.0%-(0.1%)-0.1%	0.0%-(5.5%)-100%	0.0%-(0.1%)-0.1%
	5	29.7%-(38.2%)-61.2%	0	100%-(**100%**)-100%	29.7%-(38.2%)-61.2%	45.8%-(54.9%)-76.0%
			1	0.0%-(0.1%)-0.1%	0.0%-(4.5%)-100%	0.0%-(0.1%)-0.1%
(c)	**1**	99.7%-(99.8%)-99.8%	0	99.9%-(99.9%)-100%	99.6%-(99.6%)-99.6%	99.8%-(99.8%)-99.8%
			1	99.4%-(99.6%)-99.6%	99.9%-(99.9%)-100%	99.7%-(99.8%)-99.8%
	2	84.6%-(88.1%)-88.4%	0	99.9%-(99.9%)-100%	84.6%-(88.1%)-88.4%	91.7%-(93.7%)-93.9%
			1	0.0%-(0.1%)-0.1%	0.0%-(4.7%)-40.0%	0.0%-(0.1%)-0.1%
	3	90.0%-(90.6%)-90.7%	0	99.9%-(99.9%)-100%	90.0%-(90.6%)-90.7%	94.7%-(95.1%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	4	45.2%-(83.0%)-90.7%	0	99.9%-(99.9%)-100%	45.2%-(83.0%)-90.7%	62.3%-(89.6%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%
	5	45.2%-(83.7%)-90.7%	0	99.9%-(**99.9%**)-100%	45.2%-(83.7%)-90.7%	62.3%-(90.4%)-95.1%
			1	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%	0.0%-(0.0%)-0.0%

Chapter 6

Log Message Anomaly Detection with Fuzzy C-means and MLP

The goal in this chapter is to detect anomalies in large log message data sets using only a very small amount of labeled data for training. The proposed model uses FCM and MLP for anomaly detection. First, the training data is processed using FCM with more clusters than the number of data classes. The cluster centers and a radius are employed to select reliable logs with a k-dimensional tree (kd-tree). Then, these reliable positive and negative logs are used for anomaly detection using an MLP network. Finally, class probabilities with an expert are used to correct the network output for suspect logs. The proposed model is evaluated using the accuracy, precision, recall and F-measure metrics with three log message data sets, namely BlueGene/L (BGL), Openstack and Thunderbird.

The rest of this chapter is organized as follows. In Section 6.1, the FCM and MLP architectures are presented and the proposed model is described. Experimental results are given in Section 6.2 for the three data sets along with a discussion. Finally, Section 6.3 gives some concluding remarks.

6.1 System Model

In this section, the Fuzzy C-means (FCM) and Multilayer Perceptron (MLP) architectures are introduced along with the proposed model.

6.1.1 Fuzzy C-means (FCM)

FCM is a clustering algorithm which is an extension of the K-means algorithm [6]. In FCM, samples can be assigned to more than one cluster based on membership values which are between zero and one. The sum of the cluster membership values for a given sample must sum to one. Clustering is achieved by minimizing the objective function

$$J = \sum_{i=1}^{n} \sum_{j=1}^{c} (\mu_{ij})^m (d_{ij})^2, \tag{6.1}$$

with

$$d_{ij} = \|x_i - G_j\|, \tag{6.2}$$

where $\{x_1, x_2, \ldots, x_n\}$ is the set of n samples, c is the number of clusters, μ_{ij} is the membership of the ith sample in the jth cluster, d_{ij} is the distance (similarity) between the ith sample and the jth cluster center, G_j is the jth cluster center, and m is the fuzziness index which is a value in the range $[1, \infty]$.

The jth cluster center in iteration s is calculated as

$$G_j^s = \frac{\sum_{i=1}^{n} (\mu_{ij}^s)^m x_i}{\sum_{i=1}^{n} (\mu_{ij}^s)^m}. \tag{6.3}$$

Each center is a vector of length l which is the length of a sample ($C^s = [G_j^s]_{c \times l}$).

The memberships are

$$\begin{aligned} \mu_{ij}^{s+1} &= \frac{1}{\sum_{k=1}^{c}(\frac{d_{ij}^{s}}{d_{ik}^{s}})^{(2/m-1)}} \\ &= \frac{1}{\sum_{k=1}^{c}(\frac{\|x_i - G_j^s\|}{\|x_i - G_k^s\|})^{(2/m-1)}}, \end{aligned} \tag{6.4}$$

where s is the iteration number.

The algorithm starts with random memberships and the centers and memberships are updated iteratively. The algorithm stops when $\|U^{s+1} - U^s\| <$ a threshold or a maximum number of iterations is reached, where U is the membership matrix includes all membership values for an iteration ($U^s = [\mu_{ij}^s]_{c\times n}$).

6.1.2 Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a feedforward neural network (ANN) that consists of an input layer, an output layer, and multiple hidden layers. MLP training is conducted using a loss function with backpropagation [75]. The hidden layer output is

$$h = a(Wx + b), \tag{6.5}$$

where x is the input, W is the weight matrix and b is the bias vector. The output layer output is

$$y = a(W'h + b'), \tag{6.6}$$

where W' is the weight matrix, b' is the bias vector, and a is the activation function. Fig. 6.1 shows the architecture of an MLP with an input layer, output layer, and two hidden layers. First, the weights and biases are initialized randomly and then the inputs are fed into the network (feedforward). Next, the loss function is used to compute the error between the network output and correct output. Then, the weights and biases are updated using backpropagation and an optimizer.

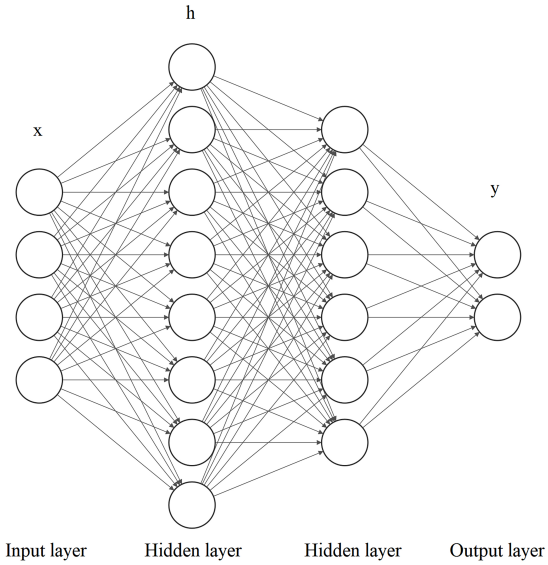

Figure 6.1: An MLP network with an input layer, output layer, and two hidden layers.

6.1.3 Proposed Model

The proposed model architecture has three steps. First, cluster centers for positive and negative logs are computed using FCM with more clusters than the number of data classes. Then radii are computed to select reliable positive and negative logs. Second, an MLP network is used with reliable logs for anomaly detection. Finally, an expert is used to correct the outputs for suspect logs using the class probabilities.

First, simple text pre-processing such as removing hyphens, changing letters to lowercase, and tokenization are applied to the data set D. Then, the logs are padded to 40 tokens, and logs with less than five tokens are eliminated. Next, the number of occurrences of each token in the data set is determined, and then the tokens are ordered from most to least frequent. Each token is given an index starting from zero and the indices are used to replace the tokens in the data set. Next, the data set D is normalized so all values are between 0 and 1 and the entries in D are shuffled. Then D is divided into two sets, a training set t_1 with 0.01% of the data and the remaining set t_2 with 99.99% of the data (0.1% and 99.9%, respectively, for the Openstack data

set). The set t_1 is small to keep the computational complexity low. The proportion of negative and positive logs in these sets is the same as in D. Then feature columns with variance less than 0.01 are removed from t_1 and t_2 and principal component analysis (PCA) is applied on t_1 and t_2.

FCM with 20 clusters, a maximum of 40 iterations, and a threshold of 0.1 with Bray-Curtis (BC) distance are used on t_1, where this distance is given by

$$BC = \frac{\sum |x_i - G_j|}{\sum |x_i + G_j|}. \tag{6.7}$$

Each log is associated with the cluster corresponding to the largest membership value in U. If the number of positive logs is greater than the number of negative logs associated with a given cluster, then this cluster is in the set C_p, otherwise it is in C_n. This maps the 20 centers to 2 classes.

Next, the means of the centers for the clusters in C_p and C_n are computed (avg_p and avg_n). Then the Euclidean distance between each cluster center and the corresponding mean is computed. The minimum of these distances is denoted the positive and negative radius, R_p and R_n, respectively. If there is only one cluster in a set, the radius for the other set is used. Then R_p and R_n are multiplied by 0.01 so that the initial radius is small.

A k-dimensional tree search (kd-tree) is used with t_2 and the positive and negative radii to select reliable positive and negative logs (re_p and re_n). A log is reliable if it lies within the corresponding radius of a cluster center (with output $z = 1$ for positive logs and $z = 0$ for negative logs). If the number of reliable logs is less than 10000 for each of the positive or negative classes, then the radius is multiplied by 2 and reliable logs are selected again with kd-tree. Then re_p and re_n are shuffled and 80% are selected for Tr_1 and 20% for v_1. The set t_3 set is obtained by removing the logs in re_p and re_n from t_2. Next, t_3 is shuffled and split with 10% for t_4 and 90% for t_5.

The MLP_1 network is trained with Tr_1 and validated with v_1. This network has two hidden layers with 128 and 64 units, respectively, and 2 units for the output layer. The maximum number of epochs for the MLP_1 network is 100 and the batch size is 64. Early stopping is applied to prevent overfitting, and the Adam optimizer and cross entropy loss function are used for training. Then the MLP_1 is tested with t_4 and the probability of a log being in each class is obtained, i.e. $prob_0$ for class 0 (negative logs) and $prob_1$ for class 1 (positive logs), from the last layer. The logs in t_4 with $|prob_0 - prob_1| < 0.5$ are selected as u_1 (class probabilities). Then the logs in u_1 are sorted from the smallest to largest values of $|prob_0 - prob_1|$ and set as u_2 (suspect logs). If the number of logs in u_2 is less than 100, then MLP_1 is tested with t_5 and the algorithm is terminated (no need for an expert). Otherwise, the first 100 logs in u_2 are selected as suspect logs and labeled as u_3. An expert is used to correct any errors in the outputs (z) of u_3. Two cases are considered. With 2% error, the expert is assumed to have made 2 mistakes so two outputs are randomly changed and the resulting 100 logs are set to u_4. With no error, the correct outputs are used in u_4. Next, Tr_1 and u_4 are concatenated and set to t_6. Then t_6 is shuffled and split with 80% for Tr_2 and 20% for v_2. Finally, a second MLP network MLP_2 is trained with Tr_2 and v_2, and tested with t_5. The parameters for MLP_2 are the same as for MLP_1. The proposed model algorithms and architecture are given in Algorithms 6-9 and Fig. 6.2.

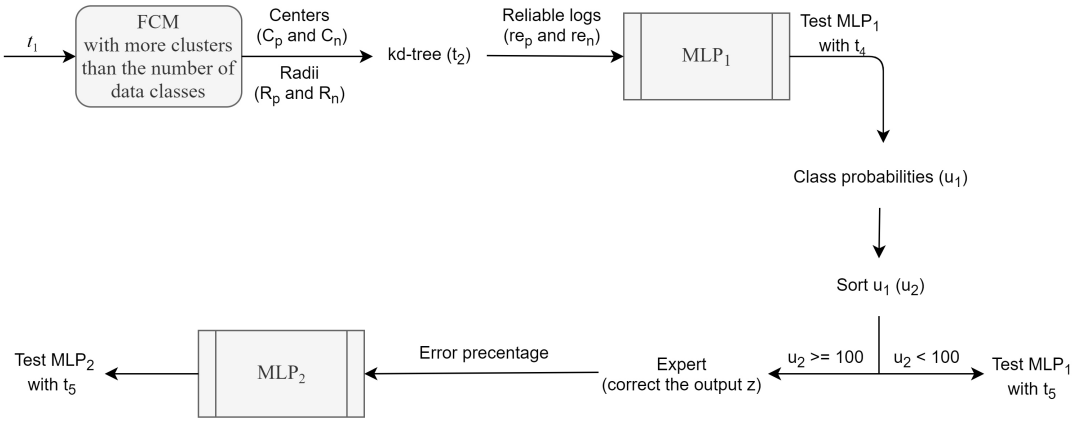

Figure 6.2: The architecture of the proposed model with FCM and MLP.

Algorithm 6 FCM and means of cluster centers

Input: Data set D scaled between $[0, 1]$, training set t_1 (0.01% of D), remaining set t_2 (99.99% of D)

1: Remove features with variance less than 0.01 from t_1 and t_2

2: Apply PCA to t_1 and t_2

3: Apply FCM to t_1 with more clusters than the number of data classes

4: Map clusters to the sets C_p and C_n according to the class of the majority of logs

5: $avg_p \leftarrow$ mean of cluster centers in C_p

6: $avg_n \leftarrow$ mean of cluster centers in C_n

Algorithm 7 Compute radii

if $|C_p| \geq 2$ **and** $|C_n| \geq 2$ **then**
    for i in C_p **do**
        Append $R_p \leftarrow \text{Euclidean}(C_p[i] - avg_p)$
    for i in C_n **do**
        Append $R_n \leftarrow \text{Euclidean}(C_n[i] - avg_n)$
else
    if $|C_p| < 2$ **then**
        for i in C_n **do**
            Append $R_n \leftarrow \text{Euclidean}(C_n[i] - avg_n)$
        $R_p = R_n$
    if $|C_n| < 2$ **then**
        for i in C_p **do**
            Append $R_p \leftarrow \text{Euclidean}(C_p[i] - avg_p)$
        $R_n = R_p$
$R_p \leftarrow \min(R_p) \times 0.01$
$R_n \leftarrow \min(R_n) \times 0.01$

Algorithm 8 Select reliable logs

re_p, $re_n \leftarrow 0$

while $|re_p| < 10000$ **do**

    $re_p \leftarrow$ kd-tree(R_p, C_p) on t_2 to select reliable positive logs

    $R_p \leftarrow R_p \times 2$

while $|re_n| < 10000$ **do**

    $re_n \leftarrow$ kd-tree(R_n, C_n) on t_2 to select reliable negative logs

    $R_n \leftarrow R_n \times 2$

Concatenate and shuffle re_p and re_n, and split with 80% for Tr_1 and 20% for v_1

$t_3 \leftarrow re_p + re_n - t_2$

Shuffle and split t_3 with 10% for t_4 and 90% for t_5

Algorithm 9 Class probabilities and anomaly detection

Train MLP_1 with Tr_1 and v_1
Test MLP_1 with t_4 and get class probabilities for each log ($prob_0$ and $prob_1$)
$u_1 \leftarrow$ logs with $|prob_0 - prob_1| < 0.5$ from t_4 (class probabilities)
$u_2 \leftarrow$ sort u_1 from smallest to largest values of $|prob_0 - prob_1|$
if $|u_2| < 100$ **then**
    Test MLP_1 with t_5
else
    $u_3 \leftarrow$ first 100 logs in u_2 with the predicted outputs (z) corrected by an expert
    if 2% error **then**
        $u_4 \leftarrow u_3$ with 2 logs outputs (z) changed
    else
        $u_4 \leftarrow u_3$
    $t_6 \leftarrow$ concatenate Tr_1 and u_4
    Shuffle and split t_6 with 80% for Tr_2 and 20% for v_2
    Train MLP_2 with Tr_2 and v_2
    Test MLP_2 with t_5

6.2 Results

The proposed model is evaluated in this section with the BGL, Openstack and Thunderbird data sets. Four performance metrics are used to evaluate, namely accuracy, precision, recall and F-measure. All experiments were run on the Compute Canada Cedar cluster with 24 CPU cores, four P100 GPUs and 125 GB of memory, and the algorithms were implemented using the Python, Pytorch* and Scikit-learn.

The hyperparameters of the model were not tuned in the experiments and default

*https://github.com/pytorch

values were used. Each experiment was repeated 10 times and the minimum, maximum and average testing accuracy, precision, recall, F-measure and time are given in the tables. Table 6.1 gives the results for the BGL, Openstack and Thunderbird data sets before the expert. Table 6.2 gives the results for the BGL data set after the expert (with and without 2% error).

6.2.1 BGL

The BlueGene/L (BGL) data set has 4,399,502 positive log messages and 348,460 negative log messages. From these, 474 logs are used for the training set t_1 and 4,747,488 for the remaining set t_2. The average testing accuracy before the expert is 99.5% with average precision, recall and F-measure of 93.2%, 99.9% and 96.5% for negative logs, and 99.9%, 99.5% and 99.8% for positive logs, respectively. The average testing accuracy after the expert (no error) is 99.7% with average precision, recall and F-measure of 96%, 99.4% and 97.7% for negative logs, and 99.9%, 99.7% and 99.8% for positive logs, respectively. The average testing accuracy after the expert (2% error) is 99.7% with average precision, recall and F-measure of 95.7%, 99.5% and 97.5% for negative logs, and 99.9%, 99.7% and 99.8% for positive logs, respectively.

The precision, recall and F-measure results for negative log messages are better than the 92%, 91% and 92%, respectively, with the Improved K-Nearest Neighbors (IKNN) algorithm [91]. The precision, recall and F-measure results for negative log messages are also better than the 83%, 99% and 91%, respectively, for SVM unsupervised learning [30], and the 82.5%, 94.7% and 88.2%, respectively, for the nLSALog algorithm [98]. Several well-known models were also evaluated. The average testing accuracy, precision, recall, F-measure and time for the BGL data set using 10-fold cross-validation with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM models are given in Table 2.1(a). The proposed model results are much better

than with these models.

6.2.2 Openstack

The Openstack data set has 137,074 positive logs and 18,434 negative logs. From these, 155 logs are used for the training set t_1 and 155,353 for the remaining set t_2. The average testing accuracy before the expert is 99.9% with average precision, recall and F-measure of 99.4%, 99.9% and 99.7% for negative logs, and 99.9%, 99.9% and 99.9% for positive logs, respectively. The precision, recall and F-measure results for negative log messages are better than the 94%, 99% and 97% with the Deeplog network [14]. Several well-known models were also evaluated. The average testing accuracy, precision, recall, F-measure and time for the Openstack data set using 10-fold cross-validation with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM models are given in Table 2.1(b). This indicates that the proposed model is much better than these other models.

6.2.3 Thunderbird

For the Thunderbird data set 3,000,000 positive log messages and 324,824 negative log messages are used. From these, 332 messages are used for the training set t_1 and 3,324,492 for the remaining set t_2. The average testing accuracy before the expert is 99.7% with average precision, recall and F-measure of 98.1%, 99.3% and 98.7% for negative logs, and 99.9%, 99.7% and 99.8% for positive logs, respectively.

The precision, recall and F-measure results for negative log messages are better than the 96%, 96% and 96% with the IKNN algorithm [91]. Also several well-known models were evaluated. The average testing accuracy, precision, recall, F-measure and time for the Thunderbird data set using 10-fold cross-validation with the BGM, EEnvelope, GMM, K-means, LOF, and OC-SVM models are given in Table 2.1(c).

This indicates that the proposed model is much better than these models.

6.2.4 Discussion

The Fuzzy C-means (FCM) algorithm has been shown to provide good results for classification tasks. However, this algorithm is not suitable for imbalanced data [55] and log messages are generally imbalanced. In this chapter, FCM is used with more clusters than the number of data classes and the cluster centers with radii are used to overcome this problem. With a greater number of clusters, their centers are more distributed which improves the selection of reliable logs. These logs are chosen based on the centers and radii.

The class probabilities allow suspect outputs to be identified after MLP prediction so an expert can check and correct the results which will improve the performance. Note that because the data set was labeled, the expert was able to correct all the errors. Thus, to be fair it is also considered the case when the expert makes an error 2% of the time, which is reasonable.

The results for the BGL, Openstack and Thunderbird data sets are shown in Table 6.1. Because the results of the Openstack and Thunderbird were very good before the expert, the number of logs in u_1 were small (less than 100). Thus, results after the expert are given only for the BGL data set in Table 6.2. In this case, results before the expert were obtained by ignoring the condition $u_2 \geq 100$. These show that using the class probabilities improves the performance with the BGL data set regardless of whether the expert makes errors. The results with and without errors were approximately the same. The average precision and F-measure are increased from 93.2% and 96.5% to 96% and 97.7%, respectively for negative logs. However, the average recall is decreased slightly from 99.9% to 99.4% for these logs. The average precision and F-measure for positive logs are the same before and after expert.

However, the average recall is increased slightly from 99.5% to 99.7%. A comparison of the correct and predicted outputs for the first 100 suspect logs in u_3 indicates that on average 55.6% of the log outputs were incorrect. This is high considering that the test accuracy was more than 99%, but these are the least reliable logs. Thus, sorting based on class probabilities can identify incorrectly predicted logs.

The proposed model is very fast for all the data sets with average times 138.1 s, 18.6 s and 531.2 s after pre-processing for the BGL, Openstack and Thunderbird data sets, respectively. This is because training set t_1 used with the FCM is very small (less than 0.1% of the data). It has been shown that FCM can be slow with a large training set [97]. Further, a fast search algorithm (kd-tree) is used to select reliable logs. Moreover, an MLP network is used for anomaly detection which is one of the fastest known DL algorithms.

6.3 Conclusion

Log messages are a valuable source of information in software systems. Hence, automated anomaly detection using these messages can save organizations time and money. In this chapter, radius-based FCM with more clusters than the number of data classes and an MLP network were employed to detect anomalies using logs. The cluster centers with a radius were used to select reliable positive and negative logs. In addition, class probabilities with an expert were introduced to correct suspect log outputs. The proposed model was evaluated with three well-known log data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this approach performs better than other well-known methods.

Table 6.1: The testing accuracy, precision, recall, F-measure, and average time (seconds) before the expert with the minimum, maximum and average (in brackets) values for the BGL, Openstack and Thunderbird data sets for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

Data set	**Testing Accuracy**	**Label**	**Precision**	**Recall**	**F-measure**	**Time (s)**
BGL	99.4%-(**99.5%**)-99.7%	0	90.6%-(**93.2%**)-95.4%	99.9%-(**99.9%**)-100%	95.1%-(**96.5%**)-97.6%	138.1
		1	99.9%-(99.9%)-100%	99.4%-(99.5%)-99.6%	99.7%-(99.8%)-99.8%	
Openstack	99.9%-(**99.9%**)-100%	0	97.8%-(**99.4%**)-100%	99.9%-(**99.9%**)-100%	98.9%-(**99.7%**)-100%	18.6
		1	99.9%-(99.9%)-100%	99.9%-(99.9%)-100%	99.9%-(99.9%)-100%	
Thunderbird	97.7%-(**99.7%**)-99.9%	0	82.2%-(**98.1%**)-99.9%	96.7%-(**99.3%**)-99.7%	88.9%-(**98.7%**)-99.8%	531.2
		1	99.6%-(99.9%)-99.9%	97.8%-(99.7%)-99.9%	98.7%-(99.8%)-99.9%	

Table 6.2: The testing accuracy, precision, recall, F-measure, and average time (seconds) after the expert with (a) no error and (b) 2% error, with the minimum, maximum and average (in brackets) values for the BGL data set for 10 runs. Positive labels are denoted by 1 and negative labels by 0.

	Testing Accuracy	**Label**	**Precision**	**Recall**	**F-measure**
No Error	99.6%-(**99.7%**)-99.8%	0	93.9%-(**96%**)-97.9%	98.8%-(**99.4%**)-100%	96.8%-(**97.7%**)-98.3%
		1	99.9%-(99.9%)-100%	99.6%-(99.7%)-99.8%	99.8%-(99.8%)-99.9%
2% Error	99.5%-(**99.7%**)-99.8%	0	91.7%-(**95.7%**)-97.7%	99%-(**99.5%**)-100%	95.7%-(**97.5%**)-98.3%
		1	99.9%-(99.9%)-100%	99.5%-(99.7%)-99.8%	99.7%-(99.8%)-99.9%

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Most organizations now use cloud or application servers. Companies and consumers demand 24/7 accessibility to their cloud and application services as connection failures can severely affect operations. Log messages that show the state of the system are employed to improve efficiency and reliability. Log message anomaly detection is a very important task that can be solved using ML. In Chapter 2, Auto-LSTM, Auto-BLSTM and Auto-GRU models were proposed for log message anomaly detection. The Autoencoder network was used to extract features from the input data and anomaly detection was done with an LSTM, BLSTM or GRU network. The proposed models were tested on three log message data sets for anomaly detection while the fourth is a sentiment movie review classification data set. The proposed models were shown to provide good results for these data sets.

In Chapter 3, a model was proposed for oversampling log messages. First, the negative logs were oversampled with a SeqGAN network so the number of positive and negative logs is similar. Then an Autoencoder was used to extract features from the text data. Finally, A GRU network was used for anomaly detection. The

proposed model was evaluated using two log data sets, namely BGL and Openstack. The results obtained indicate that oversampling improves detection and classification accuracy.

In Chapter 4, a model was proposed for unsupervised anomaly detection using Isolation Forest and two deep Autoencoder networks. The Autoencoder networks were used for feature extraction and anomaly detection, and Isolation Forest was used to predict positive data. The number of negative logs predicted to be positive with Isolation Forest was low, particularly with one Autoencoder. Thus, most logs which are predicted to be positive are correct. The proposed model was evaluated using three well-known logs data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model is better than other models.

In Chapter 5, a hybrid log message anomaly detection technique with pruning of positive and negative log messages using DL was proposed. To prune positive logs, an unsupervised algorithm with a GMM was used. Then, an unsupervised algorithm was used to prune negative logs with the K-means, GMM, and BGM methods iteratively. The precision with the pruning algorithms for positive and negative logs was high. The proposed model was tested on three different logs data sets, namely BGL, Openstack and Thunderbird. The results obtained show that this model is better than other algorithms.

In Chapter 6, radius-based FCM with more clusters than the number of data classes and an MLP network were employed to detect anomalies. The cluster centers with a radius were used to select reliable positive and negative logs. Moreover, class probabilities with an expert were introduced to correct suspect log outputs. The proposed model was evaluated with three log data sets, namely BGL, Openstack and Thunderbird. The results obtained show that the proposed model performs better than several other methods.

7.2 Future Work

PUL is a machine learning technique for one-class classification. In this technique, only positive data is used. A two-stage PUL technique was given in [49]. Positive samples are considered to be similar to the labeled samples and distinct from the negative samples. In the first stage, negative samples are identified. In stage two, a supervised method is trained with positive samples, negative samples, and the existing unlabeled samples. Then, the best classifier is chosen based on the results from the previous stage. Thus, a two stage PUL technique can be developed for log message anomaly detection when only positive logs are available.

Despite the increasing number of applications for DL, deep neural network training continues to be difficult. The barriers to training can be divided into two types, problems that prohibit a neural network from producing good performance and problems like overfitting [105]. To date, there has been no investigation of the effects of model initialization on log message anomaly detection methods. Thus, to address the issues with training, the effects of initialization methods like Glorot [20] and the role of initialization in anomaly detection can be investigated.

The performance of a neural network is affected by the architecture of the network and the activation function used in the units. Activation functions also influence the complexity and efficiency of the networks, as well as algorithm convergence [79]. Thus, the selection of an activation function has a significant impact on the overall performance of the network. Thus, a low complexity activation function can be developed specifically for anomaly detection.

Bibliography

[1] C. C. Aggarwal and C. Zhai. A Survey of Text Clustering Algorithms. In *Mining Text Data*, pages 77–128. Springer, Berlin, 2012.

[2] H. Alashwal, M. El Halaby, J. J. Crouse, A. Abdalla, and A. A. Moustafa. The Application of Unsupervised Clustering Methods to Alzheimer's Disease. *Frontiers in Computational Neuroscience*, 13(31):1–9, 2019.

[3] M. Alkhayrat, M. Aljnidi, and K. Aljoumaa. A Comparative Dimensionality Reduction Study in Telecom Customer Segmentation using Deep Learning and PCA. *Journal of Big Data*, 7(9):1–23, 2020.

[4] M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, and C. Perera. Smart Audio Sensors in the Internet of Things Edge for Anomaly Detection. *IEEE Access*, 6:67594–67610, 2018.

[5] M. Bahrololum and M. Khaleghi. Anomaly Intrusion Detection System Using Gaussian Mixture Model. In *International Conference on Convergence and Hybrid Information Technology*, pages 1162–1167, 2008.

[6] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Advanced Applications in Pattern Recognition. Springer, Berlin, 1981.

[7] D. M. Blei and M. I. Jordan. Variational Inference for Dirichlet Process Mixtures. *Bayesian Analysis*, 1(1):121–143, Mar. 2006.

[8] V. Chavan, A. Malage, K. Mehrotra, and M. K. Gupta. Printed Text Recognition using BLSTM and MDLSTM for Indian Languages. In *International Conference on Image Information Processing*, pages 1–6, 2017.

[9] N. V. Chawla. Data Mining for Imbalanced Datasets: An Overview. In *Data Mining and Knowledge Discovery Handbook*, pages 853–867. Springer, Berlin, 2005.

[10] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B Methodological*, 39(1):1–22, 1977.

[12] N. Ding, H. Ma, H. Gao, Y. Ma, and G. Tan. Real-Time Anomaly Detection Based on Long Short-Term Memory and Gaussian Mixture Model. *Computers & Electrical Engineering*, 79:106458, 2019.

[13] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017.

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

[15] J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

[16] T. S. Ferguson. A Bayesian Analysis of Some Nonparametric Problems. *The Annals of Statistics*, 1(2):209–230, 1973.

[17] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An Introduction to Deep Reinforcement Learning. *arXiv e-prints, arXiv:1811.12560*, 2018.

[18] P. Geurts, D. Ernst, and L. Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006.

[19] A. K. Ghosh, C. Michael, and M. Schatz. A Real-Time Intrusion Detection System Based on Learning Program Behavior. In *Recent Advances in Intrusion Detection*, pages 93–109. Springer, Berlin, 2000.

[20] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *International Conference on Neural Information Processing Systems*, pages 2672–2680. MIT Press, Cambridge, MA, 2014.

[23] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer, Berlin, 2012.

[24] A. Graves, S. Fernández, and J. Schmidhuber. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In W. Duch, J. Kacprzyk, E. Oja, and S. Zadrożny, editors, *Artificial Neural Networks: Formal Models and Their Applications, Lecture Notes in Computer Science*, volume 3697, pages 799–804, 2005.

[25] A. Graves and J. Schmidhuber. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5):602–610, 2005.

[26] S. Handrich, A. Herzog, A. Wolf, and C. S. Herrmann. Combining Supervised, Unsupervised, and Reinforcement Learning in a Network of Spiking Neurons. In *Advances in Cognitive Neurodynamics II*, pages 163–176. Springer, Berlin, 2011.

[27] S. Hariri, M. Carrasco Kind, and R. J. Brunner. Extended Isolation Forest. *arXiv e-prints, arXiv:1811.02141*, 2018.

[28] A. Hassan and A. Mahmood. Convolutional Recurrent Deep Learning Model for Sentence Classification. *IEEE Access*, 6:13949–13957, 2018.

[29] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang. Identifying Impactful Service System Problems Via Log Analysis. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 60–70, 2018.

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

[31] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations*, pages 1–13, 2017.

[32] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[33] J. Hofmockel and E. Sax. Isolation Forest for Anomaly Detection in Raw Vehicle Sensor Data. In *International Conference on Vehicle Technology and Intelligent Transport Systems*, pages 411–416, 2018.

[34] J. Hong and M. Y.-S. Fang. Analysis with Deeply Learned Distributed Representations of Variable Length Texts. In *Technical Report, Stanford University*, 2015.

[35] B. Hoyle, M. M. Rau, K. Paech, C. Bonnett, S. Seitz, and J. Weller. Anomaly Detection for Machine Learning Redshifts Applied to SDSS Galaxies. *Monthly Notices of the Royal Astronomical Society*, 452(4):4183–4194, 2015.

[36] E. P. Ijjina and C. K. Mohan. Hybrid Deep Neural Network Model for Human Action Recognition. *Applied Soft Computing*, 46:936–952, 2016.

[37] K. Irie, Z. Tüske, T. Alkhouli, R. Schlüter, and H. Ney. LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition. In *Interspeech*, pages 3519–3523, 2016.

[38] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep Learning for Time Series Classification: A Review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[39] C. Jiang, H.-F. Yu, C.-J. Hsieh, and K.-W. Chang. Learning word embeddings for low-resource languages by PU learning. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1024–1034, 2018.

[40] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv e-prints, arXiv:1312.6114*, 2013.

[41] T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer-Verlag, Berlin, 2001.

[42] H. Koohi and K. Kiani. User Based Collaborative Filtering Using Fuzzy C-means. *Measurement*, 91:134–139, 2016.

[43] C. Kumar, K. Rao, A. Govardhan, and K. Reddy. Imbalanced K-Means: An Algorithm to Cluster Imbalanced-Distributed Data. *International Journal of Engineering and Technical Research*, 2(2):114–122, 2014.

[44] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. Professor Forcing: A New Algorithm for Training Recurrent Networks. In *Advances in Neural Information Processing Systems*, volume 29, pages 4601–4609, 2016.

[45] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[46] D. Li, Q. Huang, X. He, L. Zhang, and M.-T. Sun. Generating Diverse and Accurate Visual Captions by Comparative Adversarial Learning. *arXiv e-prints, arXiv:1804.00861*, 2018.

[47] L. Li, R. J. Hansman, R. Palacios, and R. Welsch. Anomaly Detection Via a Gaussian Mixture Model for Flight Operation and Safety Monitoring. *Transportation Research Part C: Emerging Technologies*, 64:45–57, 2016.

[48] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen. Log Clustering Based Problem Identification for Online Service Systems. In *IEEE/ACM International Conference on Software Engineering*, pages 102–111, 2016.

[49] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu. Building Text Classifiers Using Positive and Unlabeled Examples. In *IEEE International Conference on Data Mining*, pages 179–186. IEEE Computer Society, 2003.

[50] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation Forest. In *IEEE International Conference on Data Mining*, pages 413–422, 2009.

[51] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-Based Anomaly Detection. *ACM Transactions on Knowledge Discovery from Data*, 6(1):31–39, 2012.

[52] L. Liu, Y. Lu, M. Yang, Q. Qu, J. Zhu, and H. Li. Generative Adversarial Network for Abstractive Text Summarization. *arXiv e-prints, arXiv:1711.09357*, 2017.

[53] Y.-H. Liu and Y.-T. Chen. Total Margin Based Adaptive Fuzzy Support Vector Machines for Multiview Face Recognition. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 1704–1711, 2005.

[54] M.-T. Luong, I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba. Addressing the Rare Word Problem in Neural Machine Translation. In *Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 11–19, 2015.

[55] H. Ma, C. Ekanayake, and T. K. Saha. Power Transformer Fault Diagnosis Under Measurement Originated Uncertainties. *IEEE Transactions on Dielectrics and Electrical Insulation*, 19(6):1982–1990, 2012.

[56] M. X. Ma, H. Y. T. Ngan, and W. Liu. Density-Based Outlier Detection by Local Outlier Factor on Largescale Traffic Data. In *Image Processing: Machine Vision Applications*, pages 1–4, 2016.

[57] O. Maimon and L. Rokach. Introduction to Supervised Methods. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 149–164. Springer, Berlin, 2005.

[58] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long Short Term Memory Networks for Anomaly Detection in Time Series. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 89–94, 2015.

[59] A. Maligo and S. Lacroix. Classification of Outdoor 3D Lidar Data Based on Unsupervised Gaussian Mixture Models. *IEEE Transactions on Automation Science and Engineering*, 14(1):5–16, 2017.

[60] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller. Multi-Resolution Linear Prediction Based Features for Audio Onset Detection with Bidirectional LSTM Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2164–2168, 2014.

[61] P.-F. Marteau, S. Soheily-Khah, and N. Béchet. Hybrid Isolation Forest - Application to Intrusion Detection. *arXiv e-prints, arXiv:1705.03800*, 2017.

[62] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. *arXiv e-prints, arXiv:1701.04722*, 2017.

[63] X. Miao, Y. Liu, H. Zhao, and C. Li. Distributed Online One-Class Support Vector Machine for Anomaly Detection Over Networks. *IEEE Transactions on Cybernetics*, 49(4):1475–1488, 2019.

[64] A. P. Muniyandi, R. Rajeswari, and R. Rajaram. Network Anomaly Detection by Cascading K-Means Clustering and C4.5 Decision Tree Algorithm. *Procedia Engineering*, 30:174–182, 2012.

[65] T. Munkhdalai, O.-E. Namsrai, and K. H. Ryu. Self-training in Significance Space of Support Vectors for Imbalanced Biomedical Event Data. *BMC Bioinformatics*, 16:1–8, 2015.

[66] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici. Detecting Unknown Computer Worm Activity via Support Vector Machines and Active Learning. *Pattern Analysis and Applications*, 15(4):459–475, 2012.

[67] H. Noh, T. You, J. Mun, and B. Han. Regularizing Deep Neural Networks by Noise: Its Interpretation and Optimization. In *International Conference on Neural Information Processing Systems*, pages 5115–5124, 2017.

[68] S. Otte, D. Krechel, M. Liwicki, and A. Dengel. Local Feature Based Online Mode Detection with Recurrent Neural Networks. In *International Conference on Frontiers in Handwriting Recognition*, pages 533–537, 2012.

[69] H. K. Palo, M. N. Mohanty, and M. Chandra. Use of Different Features for Emotion Recognition Using MLP Network. In *Computational Vision and Robotics*, pages 7–15. Springer, Berlin, 2015.

[70] N. Paulauskas and A. F. Bagdonas. Local Outlier Factor Use for the Network Flow Anomaly Detection. *Security and Communication Networks*, 8(18):4203–4212, 2015.

[71] T. Reidemeister, Miao Jiang, and P. A. S. Ward. Mining Unstructured Log Files for Recurrent Fault Diagnosis. In *IFIP/IEEE International Symposium on Integrated Network Management and Workshops*, pages 377–384, 2011.

[72] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*, volume 32, pages 1278–1286, 2014.

[73] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive Auto-encoders: Explicit Invariance During Feature Extraction. In *International Conference on Machine Learning*, pages 833–840, 2011.

[74] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. *arXiv e-prints, arXiv:1803.05428*, 2018.

[75] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, chapter 8: Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, 1986.

[76] H. Sak, A. Senior, and F. Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *arXiv e-prints, arXiv:1402.1128*, 2014.

[77] M. J. Siers and M. Z. Islam. Software Defect Prediction using A Cost Sensitive Decision Forest and Voting, and A Potential Solution to the Class Imbalance Problem. *Information Systems*, 51:62–71, 2015.

[78] O. Simeone. A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664, 2018.

[79] Y. Singh and P. Chandra. A Class +1 Sigmoidal Activation Functions for FFANNs. *Journal of Economic Dynamics and Control*, 28(1):183–187, 2003.

[80] G. Staerman, P. Mozharovskyi, S. Clémençon, and F. d'Alché-Buc. Functional Isolation Forest. *arXiv e-prints, arXiv:1904.04573*, 2019.

[81] Y. Sun, W. Xu, J. Zhang, J. Xiong, and G. Gui. Super-Resolution Imaging Using Convolutional Neural Networks. In *Communications, Signal Processing, and Systems*, pages 59–66. Springer, Berlin, 2020.

[82] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *International Conference on Neural Information Processing Systems*, pages 1057–1063. MIT Press, Cambridge, MA, 1999.

[83] I. Syarif, A. Prugel-Bennett, and G. Wills. Unsupervised Clustering Approach for Network Anomaly Detection. In *Networked Digital Technologies*, pages 135–145, Springer, Berlin, 2012.

[84] M. Taghavi and M. Shoaran. Hardware Complexity Analysis of Deep Neural Networks and Decision Tree Ensembles for Real-time Neural Data Classification. In *International IEEE/EMBS Conference on Neural Engineering*, pages 407–410, 2019.

[85] X. Tao, Y. Peng, F. Zhao, P. Zhao, and Y. Wang. A Parallel Algorithm for Network Traffic Anomaly Detection Based on Isolation Forest. *International Journal of Distributed Sensor Networks*, 14(11):1–11, 2018.

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

[87] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson. Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. *arXiv e-prints, arXiv:1710.00811*, 2017.

[88] R. Vaarandi, B. Blumbergs, and M. Kont. An Unsupervised Framework for Detecting Anomalous Messages from Syslog Log Files. In *IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018.

[89] T. Veracini, S. Matteoli, M. Diani, and G. Corsini. Fully Unsupervised Learning of Gaussian Mixtures for Anomaly Detection in Hyperspectral Imagery. In *International Conference on Intelligent Systems Design and Applications*, pages 596–601, 2009.

[90] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *International Conference on Machine Learning*, pages 1096–1103, 2008.

[91] B. Wang, S. Ying, G. Cheng, R. Wang, Z. Yang, and B. Dong. Log-Based Anomaly Detection with the Improved K-Nearest Neighbor. *International Journal of Software Engineering and Knowledge Engineering*, 30(2):239–262, 2020.

[92] M. Wazid and A. K. Das. An Efficient Hybrid Anomaly Detection Scheme Using K-Means Clustering for Wireless Sensor Networks. *Wireless Personal Communications*, 90(4):1971–2000, 2016.

[93] M. West and J. Harrison. *Bayesian Forecasting and Dynamic Models*, chapter Multivariate Modelling and Forecasting, pages 581–630. Springer, Berlin, 1997.

[94] M. Wöllmer, A. Metallinou, F. Eyben, B. Schuller, and S. Narayanan. Context-Sensitive Multimodal Emotion Recognition from Speech and Facial Expression Using Bidirectional LSTM Modeling. In *Interspeech*, pages 2362–2365, 2010.

[95] F. Wu, P. Anchuri, and Z. Li. Structural Event Detection from Log Messages. In *ACM International Conference on Knowledge Discovery and Data Mining*, pages 1175–1184, 2017.

[96] X. Wu, M. Klyen, K. Ito, and Z. Chen. Haiku Generation using Deep Neural Networks. In *Annual Meeting of the Natural Language Processing Society*, pages 1133–1136, 2017.

[97] Y. Xianfeng and L. Pengfei. Tailoring Fuzzy C-Means Clustering Algorithm for Big Data Using Random Sampling and Particle Swarm Optimization. *International Journal of Database Theory and Application*, 8(3):191–202, 2015.

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

[99] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda. Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In *Annual Computer Security Applications Conference*, pages 199–208, 2013.

[100] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *arXiv e-prints, arXiv:1708.02709*, 2017.

[101] L. Yu, W. Zhang, J. Wang, and Y. Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI Conference on Artificial Intelligence*, pages 2852–2858, 2017.

[102] S. Yu, X. Zhou, and C. Li. Semi-Supervised Text Classification Using Positive and Unlabeled Data. In *Conference on Advances in Intelligent IT: Active Media Technology*, pages 249–254, 2006.

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

[104] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. *arXiv e-prints, arXiv:1409.2329*, 2014.

[105] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding Deep Learning Requires Rethinking Generalization. *arXiv e-prints, arXiv:1611.03530*, 2016.

[106] C. Zhang, X. Pan, H. Li, A. Gardiner, I. Sargent, J. Hare, and P. M. Atkinson. A Hybrid MLP-CNN Classifier for Very Fine Resolution Remotely Sensed Image Classification. *Journal of Photogrammetry and Remote Sensing*, 140:133–144, 2018.

[107] D.-Q. Zhang and S.-C. Chen. A Novel Kernelized Fuzzy C-means Algorithm with Application in Medical Image Segmentation. *Artificial Intelligence in Medicine*, 32(1):37–50, 2004.

[108] M. Zhang, B. Xu, and J. Gong. An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions. In *International Conference on Mobile Ad-hoc and Sensor Networks*, pages 102–107, 2015.

[109] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu. Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling. In *International Conference on Computational Linguistics*, pages 3485–3495, 2016.

[110] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu. Tools and Benchmarks for Automated Log Parsing. In *International Conference on Software Engineering: Software Engineering in Practice*, pages 121–130, 2019.