Application of Machine Learning Techniques for Detecting Anomalies in Communication Networks

by

Qingye Ding

M.Sc., New Jersey Institute of Technology, 2014
B.Sc., Northeast Agricultural University, 2012

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Summer 2018

Approval

Name:	**Qingye Ding**
Degree:	**Master of Applied Science (Engineering Science)**
Title:	**Application of Machine Learning Techniques for Detecting Anomalies in Communication Networks**
Examining Committee:	**Chair: Ivan V. Bajić** Professor
	Ljiljana Trajković Senior Supervisor Professor
	Parvaneh Saeedi Supervisor Associate Professor
	Qianping Gu Internal Examiner Professor School of Computing Science
Date Defended:	**June 19, 2018**

Abstract

Detecting, analyzing, and defending against cyber threats is an important topic in cyber security. Applying machine learning techniques to detect such threats has received considerable attention in research literature. Anomalies of Border Gateway Protocol (BGP) affect network operations and their detection is of interest to researchers and practitioners. In this Thesis, we describe main properties of the BGP and datasets that contain BGP records collected from various public and private domain repositories such as Réseaux IP Européens (RIPE) and BCNET.

With the development of fast computing platforms, the neural network-based algorithms have proved useful in detecting BGP anomalies. We apply the Long Short-Term Memory machine learning technique for classification of known network anomalies. The models are trained and tested on various collected datasets. Various classification techniques and approaches are compared based on accuracy and F-Score.

Keywords: Border Gateway Protocol; routing anomalies; machine learning; classification algorithms; long short-term memory

Acknowledgements

I would like to express my sincere appreciation to my advisor Prof. Ljiljana Trajković for her great guidance with my research, for her patience, kindness, and immense knowledge.

I would also like to thank the rest of my committee: Prof. Ivan V. Bajić, Prof. Qianping Gu, and Prof. Parvaneh Saeedi, for their insightful comments and suggestions.

My thanks also go to my friends and colleagues in Communication Networks Laboratory at Simon Fraser University for their support and assist during my graduate study.

Finally, I would like to thank my family for supporting me spiritually throughout writing this Thesis. In particular, I am grateful to my husband for his wise counsel and sympathetic ear. You were always there for me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview of Border Gateway Protocol (BGP)

Border Gateway Protocol (BGP) [1] is a routing protocol that plays an essential role in forwarding Internet Protocol (IP) traffic between the source and the destination Autonomous Systems (ASes). An Autonomous System (AS) is a collection of BGP peers (neighbors) managed by a single administrative domain [2]. It consists of one or more networks that possess uniform routing policies while operating independently. Internet operations such as connectivity and data packet delivery are facilitated by various ASes.

The main function of BGP is to select the best routes between ASes based on network policies enforced by network administrators. Routing algorithms determine the route that a data packet takes while traversing the Internet. They exchange reachability information about possible destinations. BGP is an upgrade of the Exterior Gateway Protocol (EGP) [3]. It is an interdomain routing protocol used for routing packets in networks consisting of a large number of ASes. BGP version 4 allows Classless Interdomain Routing (CIDR), aggregation of routes, incremental additions, better filtering options, and it has the ability to set routing policies. BGP employs the Path Vector protocol, which is a modified version of the Distance Vector protocol [5]. It is a standard for the exchange of information among the Internet Service Providers (ISPs).

BGP relies on the Transport Control Protocol (TCP) to establish a connection (port 179) between the routers. A BGP router establishes a TCP connection with its peers that reside in different ASes. Because of their size, BGP routing tables are exchanged once between the peering routers when they first connect. BGP allows ASes to exchange reachability information with peering ASes to transmit information about the availability of routes within an AS. Based on the exchanged information and routing policies, it determines the most appropriate path to destination. Hence, BGP allows each subnet to announce its existence to the Internet and to publish its reachability information. Hence, all sub-networks are interconnected and are known to the Internet.

BGP is an incremental protocol that sends updates only if there are reachability or topology changes within the network. Afterwards, only updates regarding new prefixes or withdrawals of the existing prefixes are exchanged. BGP routers exchange four types of messages: *open*, *update*, *keep-alive*, and *notification* [3]. After a transport protocol connection is established, the *open* message that contains basic information such as router identifier, BGP version, and the AS number is used to open a peering session. Once the *open* message is confirmed, *update*, *keep-alive*, and *notification* messages are exchanged. BGP routers exchange known routing information using the *update* message after the BGP session is established and when there is a change of BGP routes in their routing tables. They advertise only one available route, withdraw multiple unavailable routes, or execute both simultaneously. Network Layer Reachability Information (NLRI) field of the *update* message contains attributes of all paths that apply to the destination. The *Update* message contains information of the relationships among various ASes. It may be used to detect network anomalies and prevent routing loops. The Minimal Route Advertisement Interval (MRAI) limits the minimum time interval between two update messages that are sent to the same destination. The default MRAI value is 30 s in practice [4]. Instead of using transport protocol to determine if peers are reachable, BGP routers frequently exchanges *keep-alive* messages between peers during inactivity periods to ensure that the connections do not expire the hold time, which is usually 90 seconds. The maximum interval between *keep-alive* messages is typically one third of the *hold time* interval. The *notification* message closes a peering session if there is a disagreement in the configuration parameters.

A sample of a BGP update message is shown in Table 1.1. The *Origin* attribute is generated by the AS that originates the routing information. It is a mandatory attribute and will be propagated to other BGP speakers that are used to advertise the routing information. The *AS-path* attribute in the BGP update message indicates the path that a BGP packet traverses among AS peers. Only when a BGP speaker propagates a route to another BGP speaker located within the same AS, the *AS-path* attribute will not be modified. The *AS-path* attribute enables BGP to route packets via the best path. The *next-hop* attribute defines the IP address of the next hop to the destination. NLRI announcements share a list of IP address prefixes.

Propagation of the BGP routing information is susceptible to various anomalous events such as worms, malicious attacks, power outages, blackouts, and misconfigurations of BGP routers. BGP anomalies are caused by changes in network topologies, updated AS policies, or router misconfigurations. They affect the Internet servers and hosts and are manifested by anomalous traffic behavior. Anomalous events in communication networks cause traffic behavior to deviate from its usual profile. These events may spread false routing information throughout the Internet by either dropping packets or directing traffic through unauthorized ASes and, hence, risking eavesdropping. Large-scale power outages may affect ISPs due to unreliable power backup. They could also cause network equipment failures leaving affected

Table 1.1: Sample of a BGP update message. IGP: Interior Gateway Protocol; NLRI: Network layer reachability information.

Field	**Value**
TIME	2003 1 24 00:39:53
TYPE	BGP4MP/BGP4MP_MESSAGE AFI_IP
FROM	192.65.184.3
TO	193.0.4.28
BGP PACKET TYPE	UPDATE
ORIGIN	IGP
AS-PATH	513 3320 7176 15570 7246 7246 7246
	7246 7246 7246 7246 7246 7246
NEXT-HOP	192.65.184.3
ANNOUNCED NLRI PREFIX	198.155.189.0/24
ANNOUNCED NLRI PREFIX	198.155.241.0/24

networks isolated and their service disrupted. Configuration errors in BGP routers also induce anomalous routing behavior. Routing table leak and prefix hijack [6] events are examples of BGP configuration errors that may lead to large-scale disconnections in the Internet. A routing table leak occurs when an AS (such as an ISP) announces a prefix from its Route Information Base (RIB) that violates previously agreed upon routing policy. A prefix hijack is the consequence of an AS originating a prefix that it does not own.

1.2 Machine Learning Techniques

Machine learning is a subfield of artificial intelligence, which is closely related to statistics and various interdisciplinary fields, such as neocognitron in biology [7], cognitive science, control theory, and information theory [8]. Unlike traditional computing techniques, machine learning enables computers to build models and make decisions based on input datasets. Machine learning was defined [9] as a computer program that could learn from experiences with respect to classes of objectives and performance metrics, as long as they improve the experience.

The concept of machine learning was introduced in Alan Touring's paper "Computing Machinery and Intelligence" [10]. Machine learning gained more attention after he brought up the research question "Can machines think?" in 1950. Since 1990s, machine learning flourished and became well known. In 2006, Geoffrey Hinton introduced the deep learning and proved that machines are able to distinguish objects and texts in images and videos [11]. Machine learning is able to analyze complex and large volume of data, which makes it crucial in various areas. For example, image recognition technology allows surveillance cameras to detect faces; speech recognition translates spoken words into texts while recommender systems suggest commodities based on user preferences. Machine learning techniques have

also been employed to develop models for detecting anomalies and designing BGP anomaly detection systems [12]. They are the most common approaches for classifying BGP anomalies. A typical goal of machine learning applications is to map input instances to an output value. An illustration of the machine learning process is shown in Fig. 1.1.

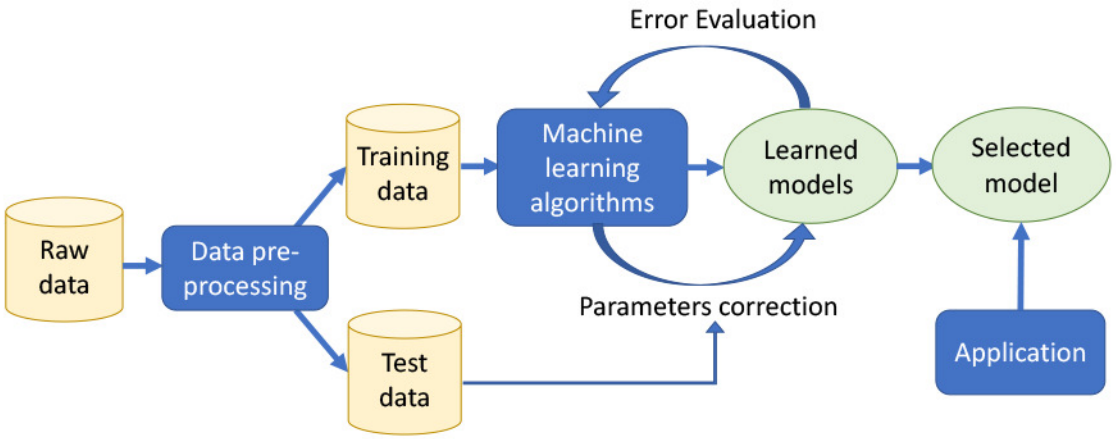

Figure 1.1: Illustration of the machine learning process.

1.2.1 Machine Learning Process

Raw data typically contain missing values and redundant information and, thus, they need to be parsed, cleaned, pre-processed, and transformed into a proper form. The data is then divided into training and test datasets. In the training phase, the machine learning algorithms are applied to the training data to create candidate models. Models "observe" and learn from the training dataset. However, a model may be biased due to the limited number of data points that the model has observed. In order to avoid such biased observations, the validation phase is introduced to estimate the quality of the model based on classification accuracy, recall, precision, or F-Score. Hyperparameters of the model are tuned based on the validation results to improve the existing model.

K-fold cross-validation is commonly used in machine learning tasks [13]. The advantage of using k-fold cross validation is that the training dataset is fully used for both training and validating, and each subset is used for validation only once. In the k-fold cross-validation, the original training dataset is randomly partitioned into k equal-sized subsets. For example, for a 10-fold validation set, machine first partitions the training dataset into 10 subsets and then randomly shuffles them. Nine subsets are trained and one is left out as a validation set to estimate the quality of the model. This process repeats until all subsets have been used as both training and validation sets and have returned 10 estimation results. The average

number of the 10 results is the final estimation of the model. An illustration of a 10-fold cross validation is shown in Fig. 1.2.

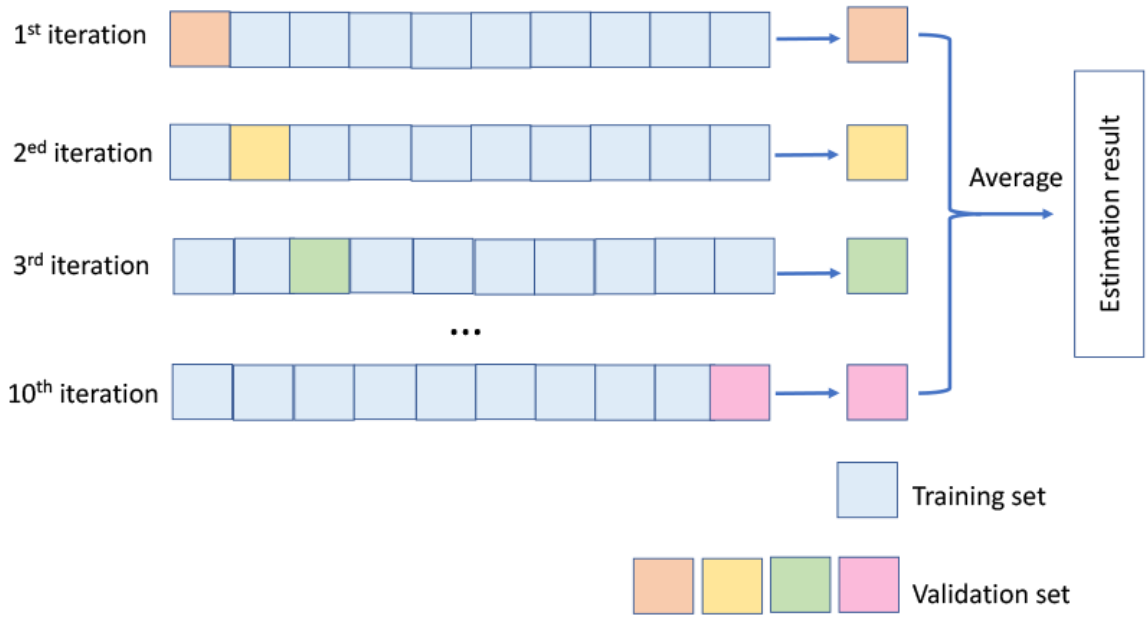

Figure 1.2: Illustration of the 10-fold cross validation. The original training dataset is partitioned into 10 folds. Each fold is used once as a validation set during the training phase. The final estimation result is the average number of the 10 validation results.

The validation phase is also used to prevent underfitting and overfitting. If the training and validation accuracy are both low, the model is underfitting, and it is not suitable for training datasets. The solution may be to use alternative machine learning algorithms or increase the number of iterations. In practice, underfitting is easy to detect while overfitting is rather challenging. If the training accuracy is high while the validation accuracy is very low, the model is probably overfitting, which implies that the model learns too well details including noise information in the training dataset and it may fail to reliably fit future observations [14]. The overfitting model learns the noise data as the useful information and captures details of the dataset. However, the capture does not suitable for new dataset and negatively impacts the generality of the model. Overfitting is more likely to happen when training nonparametric (the complexity of the model increases with the number of the training data) or nonlinear models that have distribution-free structures. Therefore, machine learning models for nonparametric datasets may use techniques to constrain the information that the model may learn. For example, a Long Short-Term Memory nonparametric model addresses overfitting by using dropout technique [15] by removing a certain portion of information that it has learned. The validation phase repeats until the validation accuracy becomes constant and is within a certain threshold. In most cases, the validation error is larger than the training error. Empirically, the training and validation sets within the

original training dataset are split in proportions 7:3 or 8:2. How to split the dataset mainly depends on the total number of data samples and properties of the training model. For example, in case of a model that needs a large number of data to train, the validation subset may be reduced to optimize the performance.

After the validation phase, test data are used to evaluate the developed models. The difference between validation and test sets is that the validation set is partitioned from the training set while the test set is only released when the training and validation phases are completed. The test set should not be learned beforehand. Otherwise, the learning is invalid. Generally the test dataset contains data samples with various classes that the model would encounter the reality. The model is then applied to the real-world data after the procedure of training, validation, and test phases.

There are various types of machine learning algorithms. They may be grouped as unsupervised or supervised learning based on their characteristics. A key distinction between unsupervised and supervised learning techniques is the target (label) [8]. For unsupervised learning, the task is to find the relationships among various unlabeled inputs. For supervised learning, every training input has a corresponding target (label) and the machine provides labels for new inputs after sufficient training.

1.2.2 Unsupervised Learning

Unsupervised learning aims to learn a function that represents the underlying structure from unlabeled input data by reducing amount or dimension of input datasets and tuning a set of parameters. The main motivation for using unsupervised learning is the labeled data is difficult to obtain, limited in quantity, and may contain errors. Unsupervised machine learning models have been used to detect anomalies in networks with non-stationary traffic [16]. The one-class neighbor machines [17] and recursive kernel-based [18] online anomaly detection algorithms are effective methods for detecting anomalous network traffic [19]. An online algorithm does not need to "observe" the entire input data and it may provide partial solutions at each iteration. In contrast, all offline algorithm requires apriori to have the entire data in order to produce the ultimate solution of the problem.

Data clustering is a commonly-used technique of unsupervised learning. Data points are grouped together based on their closeness in a suitable feature space. The goal of data clustering is to assign data points with similar traits into the same group. The procedure of data clustering typically includes feature extraction/selection, pattern representation, pattern proximity, and clustering. The procedure is illustrated in Fig. 1.3. Feature extraction technique transforms the original input data and extracts desired features. Feature selection is the process where the system selects the most effective features from the original dataset. These techniques may be used in clustering to obtain the most suitable feature sets. Pattern representation refers to the information extracted or selected from the original dataset such

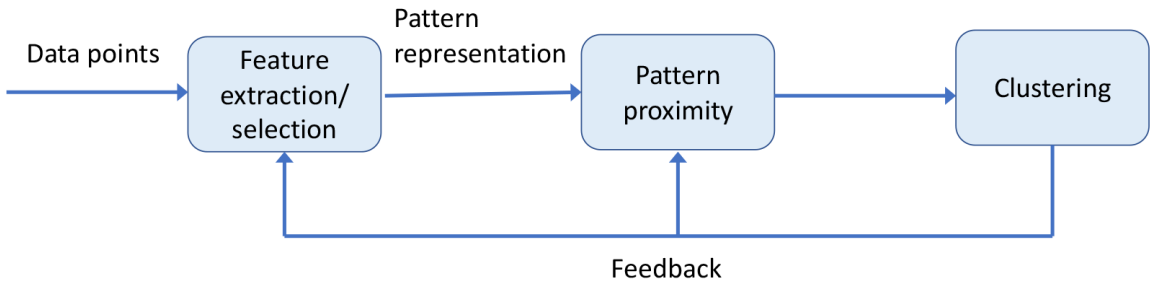

Figure 1.3: Procedure of data clustering with a feedback loop. The result of clustering may affect feature extraction/selection and the computation of pattern proximity.

as the number of classes, types, and the number of available patterns. Pattern proximity basically forms the elements in a group when they are placed close to each other.

Distance functions are used to measure distances between pairs of patterns. The measurement of distance should be carefully selected because the feature types and scales vary in a dataset. The Euclidean distance is the most commonly used metric to evaluate the proximity of patterns. It is well suited for datasets with isolated clusters. The Euclidean distance [20] between two data points is the length of the straight line between them. It is calculated as:

$$\begin{aligned} dist_{Euc}(x_i, x_j) &= \sqrt{(\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2)} \\ &= ||x_i - x_j||_2, \end{aligned} \tag{1.1}$$

where x_i, x_j, and x_k are data points in a d-dimensional feature space, where x_i has coordinates $x_{ik}, \cdots, x_{id}$ and x_j has coordinates $x_{jk}, \cdots, x_{jd}$. Euclidean distance is a special case that is only suitable for data points in two or three dimensional space. A general form of Euclidean distance is the Minkowski distance [21]. For higher dimensional feature space d, Minkowski metric uses the L_p norm that is calculated as:

$$\begin{aligned} dist_{p}(x_i, x_j) &= \sqrt[p]{(\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^p)} \\ &= ||x_i - x_j||_p, \end{aligned} \tag{1.2}$$

Another well-known metric is Cosine distance. It measures the divergence between a pair of data points using the cosine of the angle between their locations. The Cosine distance is well suited for high-dimensional data. Thus, it is widely used in data mining for tasks with large datasets such as identifying plagiarism. The Cosine distance is calculated as:

$$\begin{aligned} dist_{cos}(x_i, x_j) &= 1 - cos(\widehat{x_i, x_j}) \\ &= 1 - \frac{\sum_{k=1}^{d}(x_{ik} \cdot x_{jk})}{\sqrt{\sum_{k=1}^{d}(x_{ik})^2}\sqrt{\sum_{k=1}^{d}(x_{jk})^2}} \end{aligned} \tag{1.3}$$

After the pattern proximity is identified, clustering algorithms are used. It is the key of the data clustering procedure. Clustering techniques may be divided into several categories.

Hard vs. fuzzy clustering techniques: Clustering techiniques may be categorized as hard or fuzzy (soft) based on the clustering output. A hard clustering algorithm assigns each data point to a distinct cluster and each data point may only belong to one cluster. Instead of assigning data points to distinct clusters, a fuzzy clustering (also called soft clustering) may assign each data point to multiple clusters. Membership grade is used in the soft clustering to maintain a list of cluster nodes without inappropriate duplications. The list continuously changes during the clustering procedure. Based on the membership list, soft clustering algorithms assign each data point a likelihood to indicate the probability of a data to be allocated in different clusters. An examples of hard clustering method is shown in Fig. 1.4. An illustration of fuzzy clustering is shown in Fig. 1.5. Each data point in the original dataset is assigned a membership degree for each cluster that it may be allocated. For instance, data point 1 (a) has membership degrees $u_{1,1}$, $u_{1,2}$, and $u_{1,3}$ corresponding to its potential clusters 1, 2, and 3, respectively. The final clustering result (b) is generated based on membership degrees of all data points.

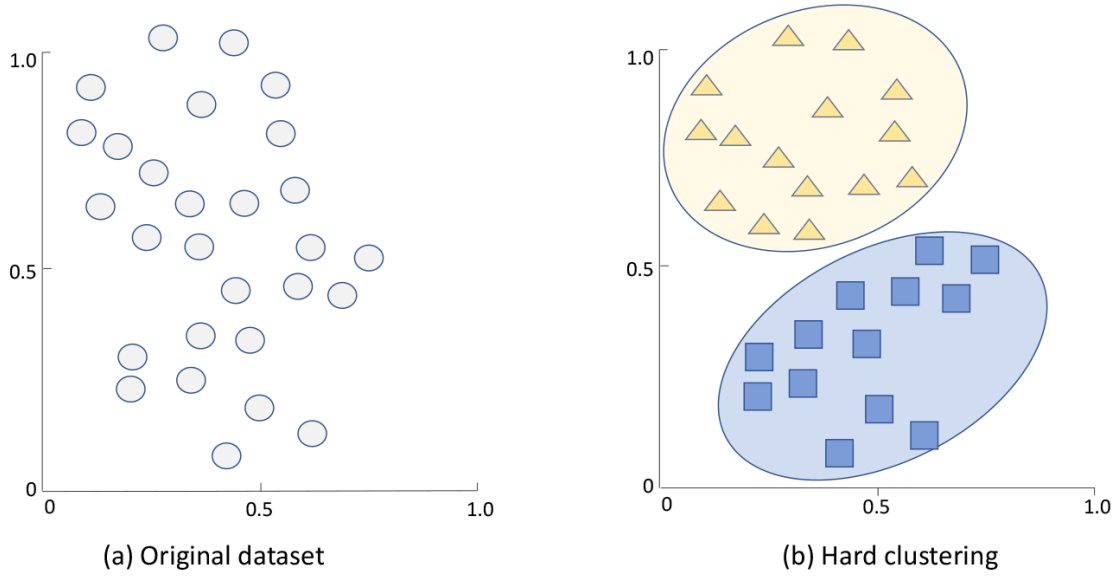

Figure 1.4: Illustration of hard clustering in a 2-dimensional space. The vertical and horizontal axis represent two dimensions. Each data point in the original dataset (a) is clustered to a single cluster (b). Triangle and circle represent features of the dataset.

Hierarchical vs. partitional clustering techniques: The goal of hierarchical clustering is to group data into a hierarchy or a tree of clusters. Hierarchical clustering algorithms initially assume that each data point is a cluster. The algorithms then compute the proximity (similarity between clusters) and combine pairs of the most similar clusters. The procedure

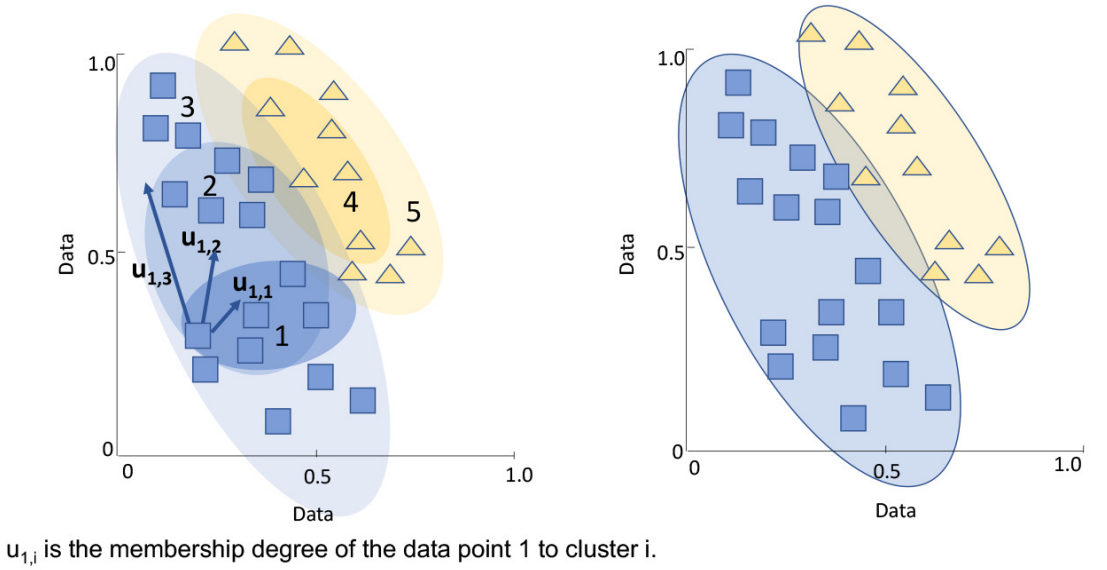

Figure 1.5: Shown is an example of fuzzy clustering in a 2-dimensional space. The vertical and horizontal axis represent two dimensions. Each data point in the original dataset (a) may belong to multiple clusters (b).

continues until reaching a pre-defined number of clusters. Unlike hierarchical methods, partitional clustering techniques do not initialize each point as a cluster. Instead, they randomly select some of the points as centers of clusters. The center point is called the centroid. Each data point is randomly assigned to a cluster. The algorithms then compute the proximity between each center and data point, update the centroid, and re-assign data points to the cluster with the most similar centroid. The procedure repeats until the number of clusters reaches a pre-defined threshold. Examples of hierarchical and partitional clustering methods are shown in Fig. 1.6 and Fig. 1.7.

Among clustering algorithms, k-means, which belongs to the partitional clustering method, has been widely used in machine learning tasks such as Natural Language Programming [8]. The number of clusters is decided apriori based on mixture models, a hierarchical clustering algorithm, or a neural network-based method such as the Kohonen map [22].

The ultimate goal of clustering is to find the most appropriate partitioning that fits the underlying data. To evaluate the performance of clustering algorithms, various clustering validation techniques have been used [23]. They may be classified into three approaches: external measurement, internal measurement, and relative criteria. External measurement evaluates the clustering results based on pre-specified ground truth that is not contained in the dataset such as structures of underlying dataset and labels. Well known metrics of external measurement are F-Score, purity, and entropy. Internal measurement validates the

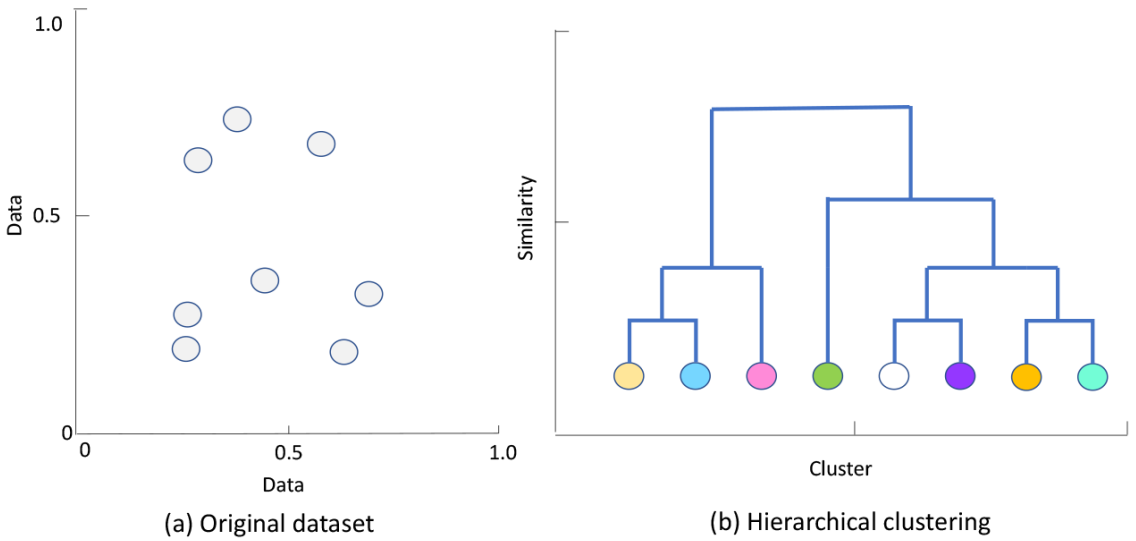

Figure 1.6: Example of hierarchical clustering. The vertical axis represents the similarity between clusters and the horizontal axis represents the combined clusters. Each data point in the original dataset (a) is considered as a single cluster (b). These clusters are merged based on their similarity.

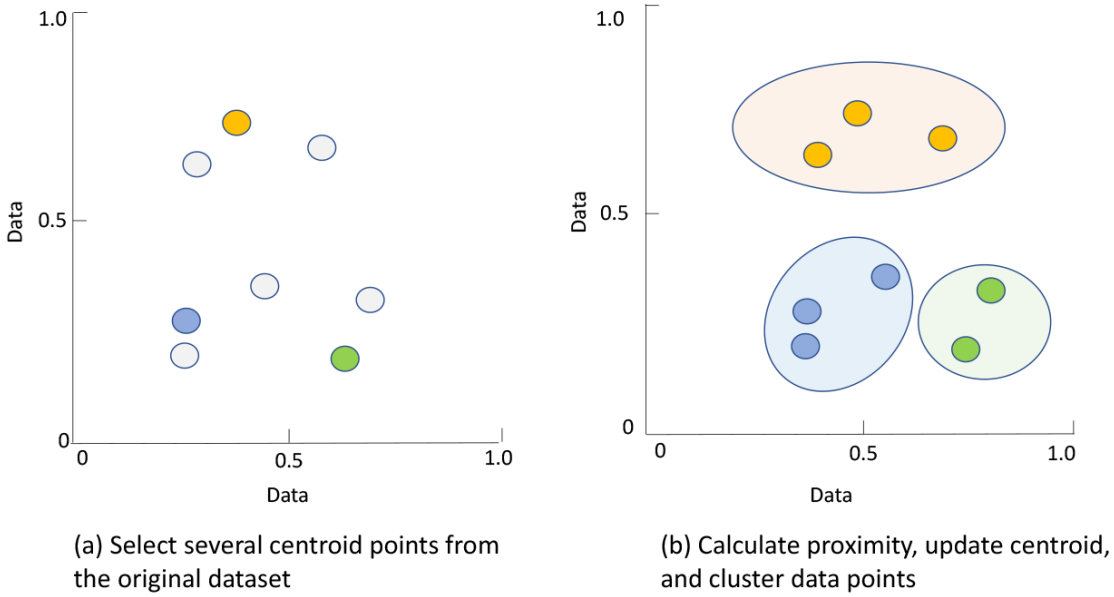

Figure 1.7: Example of partitional clustering. The vertical and horizontal axis represent two dimensions. Several centroid points are selected from the original dataset (a). The algorithm then calculates proximity of clusters (b). Centroid points and clusters are updated based on the proximity (right).

properties of created clusters in order to determine if the data points are well partitioned. These properties may be the distance between clusters, intra-cluster distance, or distances among certain points that belong to different clusters. Metrics of internal measurement [24] include Bayesian information criterion, Calinski-Harabasz index, and Dunn index. The major drawback of external and internal measurement is high computational demands. Thus, relative criteria [25] is introduced to avoid statistical tests. The approach of relative criteria is to compare a set of defined cluster schemes and select the one that best fits the pre-specified criterion.

1.2.3 Supervised Learning

Supervised learning encompasses most of the traditional types of machine learning. Unlike unsupervised learning, supervised learning obtains information and builds predictions from observed input data. In supervised learning, a labeled dataset with a specific predictive goal is given, where each observation from the dataset associates with a label corresponding to the predictive goal. A learning algorithm or classifier trains data based on the observation to predict the label for new events [8]. An example is detecting email spam, where supervised learning aims to accurately detect whether a newly received email is spam by training a large number of past emails labeled as either spam or not spam [26].

We denote a labeled training dataset as:

$$S = (X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n), \quad X_i \in \Omega, Y_i \in \{-1, 1\}, \tag{1.4}$$

where the i'th observation X_i is in a feature space Ω. We use m features/dimensions in this Thesis. Hence, $X \in \mathbb{R}^m$ and $\Omega = \mathbb{R}^m$. The label of X_i is denoted by Y_i. For binary classification problems, we use two labels: -1 for regular and 1 for anomaly. The goal of supervised learning is to train a function that maps a new observation $X \in \Omega$ to a label $Y \in \{-1, 1\}$:

$$f(X; \theta) \mapsto \{-1, 1\}, \tag{1.5}$$

where vector θ represents a set of parameters optimized by the classification algorithm to enhance the accuracy of the classifier.

Most applications employ features for classification tasks. If the function is very simple, such as a linear function $f(X; \theta) = \theta^\top X$, the classification results may only fulfill simple tasks by using few parameters. Furthermore, if the training dataset is relatively small, it is difficult to train a complex function with many parameters due to the risk of overfitting.

Supervised learning is employed for anomaly classification when the input data are labeled based on various categories. Well-known supervised learning algorithms include Long Short-Term Memory (LSTM) [27], [28], Support Vector Machine (SVM) [8], [29], Hidden Markov Model (HMM) [8], Naïve Bayes [30], Decision Tree [31], and Extreme Learning Machine (ELM) [32], [33]. The SVM algorithms often achieve better performance compared

to other machine learning algorithms albeit with high computational complexity. LSTM is trained using gradient-based learning algorithms implemented as a recurrent neural network. It outperforms other sequence learning algorithms because of its ability to learn long-term dependencies from past experiences. No single learning algorithm performs the best for all given classification tasks [34]. Hence, an appropriate algorithm needs to be selected by evaluating its performance based on various parameters. Statistical methods, data mining, and machine learning have been employed to evaluate and compare various algorithms [35], [36].

1.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are artificial neural network models that enable networks to deal with sequence recognition, pattern classification, and temporal prediction tasks with sequential inputs and outputs by performing temporal processing and sequence learning [37]. Basic RNNs consist of neuron-like nodes that directly connects to any other node. An RNN contains three types of elements: input node, hidden state, and output node. The input node receives input data from various sources and passes the information to the hidden state. The hidden state may be considered as the memory of the network. The entire memory calculated from previous time steps is reserved in the current hidden state. In practice, a traditional RNN may only capture information from limited previous time steps. The output node shows the prediction result that is calculated based on the hidden state. Across the entire learning procedure, a RNN shares the same parameters such as weights for input node, hidden state, and output node, respectively. It implies that the network is performing the same at each time step with different inputs, which reduces the total number of parameters that need to be learned. Although RNNs perform the same computation for each input element, various values of hidden states lead to different outcomes of the outputs. A typical architecture of an RNN is shown in Fig. 1.8.

At time t, the hidden state h_t is calculated based on the input x_t at the current time and the previous hidden state h_{t-1}:

$$h_t = f(W_i x_t + W_h h_{t-1} + b_h), \tag{1.6}$$

where f denotes an activation function that updates at each time step $t = 1, 2, \cdots$. The function is typically a nonlinear function such as *tanh* or Rectified Linear Unit (ReLU). W_i is the matrix of conventional weights between the input and the hidden layers while W_h is the matrix of recurrent weights between the hidden layers at adjacent time steps. The vector b_h is a bias parameter that is added to the hidden layers. The bias allows the hidden layer to generalize beyond the original dataset.

The output at time t is calculated as:

$$o_t = f(W_o h_t + b_o), \tag{1.7}$$

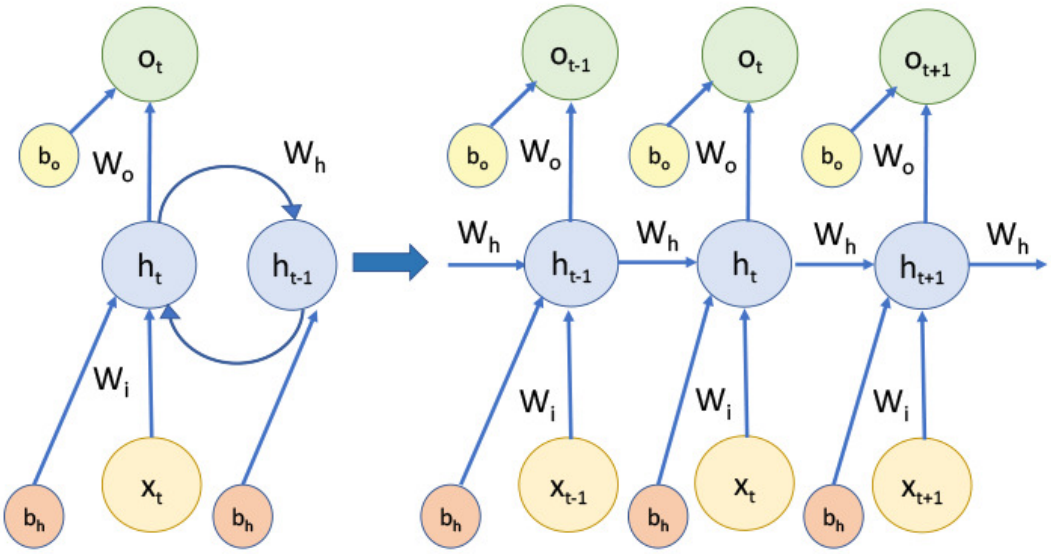

Figure 1.8: Example of a recurrent neural network (RNN). A node represents a unit. An RNN is unrolled by expanding its computation graph to a directed acyclic graph. Shown is an expanded RNN at times $t-1$, t, and $t+1$.

where W_o denotes the matrix of conventional weights between the hidden and output layers. The bias parameter is denoted by b_o.

One of the most successful RNN architectures for sequence learning tasks is Long Short-Term Memory (LSTM) [28]. It introduces the memory cell which replaces traditional nodes in the hidden layer. The memory cells enable neural networks to overcome the problem of vanishing gradients that was encountered by earlier RNNs. Another variant of LSTM is the deep LSTM architecture [38] that has multiple LSTM layers to further increase the classification accuracy. However, deep networks require larger amount of training data points.

A new direction in applying RNN architectures for anomalies detection is to use Gated Recurrent Unit (GRU) [39], which is a simplified LSTM, and, thus it is computationally more efficient than LSTMs and may require smaller training datasets. Another successful RNN is Bidirectional Recurrent Neural Network (BRNN) [40]. Unlike the traditional RNNs that only calculate outputs based on the past memory, BRNN involves both the past and the future information to predict the output sequence. BRNN has been proved to be well-suited for sequence labeling tasks in natural language processing.

1.4 Motivation

The Internet is a critical asset of information and communication technology. Border Gateway Protocol (BGP) plays an essential role in routing data between Autonomous Systems (ASes) where an AS is a collection of BGP peers administrated by a single administrative domain [41]. The main function of BGP is to select the best routes between ASes based on routing algorithms and network policies enforced by network administrators. BGP anomalies may be caused by changes in network topologies, updated AS policies, or router misconfigurations. BGP anomalies affect Internet servers and hosts and are manifested by anomalous traffic behavior. Hence, detecting such network anomalies is of great interest to researchers and practitioners.

1.5 Related Work

Detailed comparison of various network intrusion techniques has been reported in the literature [42]. Demands for Internet services have been steadily increasing and anomalous events and their effects have dire economic consequences. Determining the anomalous events and their causes is an important step in assessing loss of data by anomalous routing. Hence, it is important to classify these anomalous events and prevent their effects on BGP.

Anomaly detection techniques have been applied in communication networks [19]. These techniques are employed to detect BGP anomalies such as intrusion attacks, worms, and distributed denial of service attacks (DDoS) [43], [44] that frequently affect the Internet and its applications. BGP data have been analyzed to identify anomalous events and design tools that have been used in anomaly predictions [44–49]. Network anomalies are detected by analyzing collected traffic data and generating various classification models. A variety of techniques have been proposed to detect BGP anomalies.

Early approaches include developing traffic models using statistical signal processing techniques where a baseline profile of network regular operation is developed based on a parametric model of traffic behavior and a large collection of traffic samples to account for regular (anomaly-free) cases [45]. Anomalies may then be detected as sudden changes in the mean values of variables describing the baseline model. However, it is infeasible to acquire datasets that include all possible cases. In a network with quasi-stationary traffic, statistical signal processing methods have been employed to detect anomalies as correlated abrupt changes in network traffic [46].

The main focus of other approaches also proposed in the past is developing models for classification of anomalies. The accuracy of a classifier depends on the extracted features, combination of selected features, and underlying models. Recent research reports describe a number of applicable classification techniques [47–49]. One of the most common approaches is based on a statistical pattern recognition model implemented as an anomaly classifier and a detector [47]. Its main disadvantage is the difficulty in estimating distributions of higher di-

mensions. For example, a Bayesian detection algorithm was designed to identify unexpected route mis-configurations as statistical anomalies [48]. An instance-learning framework employed wavelets to systematically identify anomalous BGP route advertisements [49]. Other proposed techniques are rule-based methods that have been employed for detecting BGP anomalies. An example is the Internet Routing Forensics (IRF) that was applied to classify anomaly events [50]. However, rule-based techniques are not adaptable learning mechanisms. They are slow, have high degree of computational complexity, and require a priority knowledge of network conditions.

Recent trends in designing BGP anomaly detection systems more frequently rely on machine learning techniques. Known classifiers are tested for their ability to reliably detect network anomalies in datasets that include known BGP anomalies. A survey of various methods, systems, and tools used for detecting network anomalies reviews a variety of existing approaches [42]. The authors have examined recent techniques to detect network anomalies and discussed detection strategy and employed datasets, including performance metrics for evaluating detection method and description of various datasets and their taxonomy. They also identified issues and challenges in developing new anomaly detection methods and systems.

Various machine learning techniques to detect cyber threats have been reported in the literature. One-class SVM classifier with a modified kernel function was employed [51] to detect anomalies in IP records. However, unlike the approach in our studies, the classifier is unable to indicate the specific type of anomalies. Stacked LSTM networks [52] with several fully connected LSTM layers have been used for anomaly detection in time series. The method was applied to detect anomalous regions for medical electrocardiograms, space shuttle Marotta's valve time series , power demand, and engine sensors that measure dependent variables such as temperature and torque. The analyzed data contain both long-term and short-term temporal dependencies. Another example is the multi-scale LSTM [53] that was used to detect BGP anomalies using accuracy as a performance measure. In the preprocessing phase, data were compressed using various time scales. An optimal size of the sliding window was then used to determine time scale in order to achieve the best performance of the classifier. Multiple HMM classifiers [54] were employed to detect Hypertext Transfer Protocol (HTTP) payload-based anomalies for various Web applications. Authors first treated payload as a sequence of bytes and then extracted features using a sliding window to reduce the computational complexity. HMM classifiers were then combined to classify network intrusions. It was shown [55] that the Naïve Bayes classifier performs well for categorizing the Internet traffic emanating from various applications. Weighted ELM [56] deals with unbalanced data by assigning relatively larger weights to the input data arising from a minority class. Signature-based and statistics-based detection methods also have been proposed [57].

1.6 Research Contribution

In this Thesis, we consider BGP update messages because they contain the information about the protocol status and configurations. BGP update messages are extracted from the collected data during the time periods when the Internet experienced known anomalies. However, redundancies in the collected data may affect the performance of classification methods. Feature extractions are used to select a subset of features from the original feature space and, thus, reduce redundancy among features and improve the classification accuracy. Various methods for feature extraction such as principal component analysis project the original data points onto a lower dimensional space. However, features transformed by feature extraction lose their original physical meaning. We extract *AS-path* and *volume* BGP features based on the attributes of BGP update messages. We compare the performance of anomalies prediction use both unbalanced and balanced datasets.

We view anomaly detection as a classification problem of assigning an "anomaly" or "regular" label to a data point. We apply Long Short-Term Memory machine learning techniques to develop classification models for detecting BGP anomalies. These models are trained and tested using various datasets that consist extracted features. They are also used to evaluate the effectiveness of the extracted features.

The previously reported results [12] were generated using the PyBrain [58] tool. Since Keras offers additional functions such as look_back and dropout, it is used to generate results reported in this Thesis. We have also carefully processed the considered datasets and selected better parameters, which further improved classification results reported in our previous studies [12], [60], and [61].

1.7 Organization of the Thesis

The Thesis is organized as follows: We first briefly describe BGP, the effect of network anomalies and approaches for their detection. In Chapter 2, we provide details of various BGP anomalies that have been considered in this Thesis as well as the description of datasets and the data processing. Various approaches to extract features from BGP update messages are described in Chapter 3. Performance metrics and the Long Short-Term Memory algorithm are introduced in Chapter 4. We introduce various classification algorithms and compare their classification results with the LSTM algorithm in Chapter 5. The challenges of detecting BGP anomalies as well as advantages and shortcomings of various classification algorithms are offered in Chapter 6. We describe the future work and conclude in Chapter 7. The list of references is provided in the Reference section.

Chapter 2

Border Gateway Protocol Datasets

2.1 Examples of BGP Anomalies

Anomalous events considered in this Chapter are worms. They are manifested by sharp and sustained increases in the number of announcement or withdrawal messages exchanged by BGP routers. *Volume* and *AS-path* features are collected over one-minute time intervals during five-day periods for well known anomalous Internet events. While the available datasets contain data over much longer periods of time, we have selected for our analysis a five-day period to minimize storage and computational requirements. Furthermore, selecting longer periods of regular data would make datasets ever more unbalanced. Several methods that we surveyed offer better performance when dealing with balanced datasets. Details including dates of the events, remote route collectors (RRC) that acquired data using Routing Information Service (RIS), and observed peers are given in Table 2.1. For example, Slammer event occurred on January 25, 2003 and lasted almost 16 hours. Hence, BGP update messages collected between January 23, 2003 and January 27, 2003 are selected as samples for feature extraction.

Table 2.1: Examples of known BGP Internet worms.

Dataset	**Class**	**Date**		**Duration**
		Beginning of the event	End of the event	**(min)**
Slammer	Anomaly	25.01.2003 at 5:31 GMT	25.01.2003 at 19:59 GMT	869
Nimda	Anomaly	18.09.2001 at 13:19 GMT	20.09.2001 at 23:59 GMT	3,521
Code Red I	Anomaly	19.07.2001 at 13:20 GMT	19.07.2001 at 23:19 GMT	600

Records of three BGP anomalies and regular RIPE traffic are shown in Fig. 2.1–Fig. 2.4. The effect of Slammer worm on *volume* and *AS-path* features is shown in Fig. 2.5 and Fig. 2.6. We consider Slammer [62], Nimda [63], and Code Red I [64] BGP anomalous events, other Internet anomalies are also listed in table 2.2.

Slammer [62]: The Structured Query Language (SQL) Slammer worm attacked Microsoft SQL servers on January 25, 2003 [62]. Microsoft SQL servers were infected through

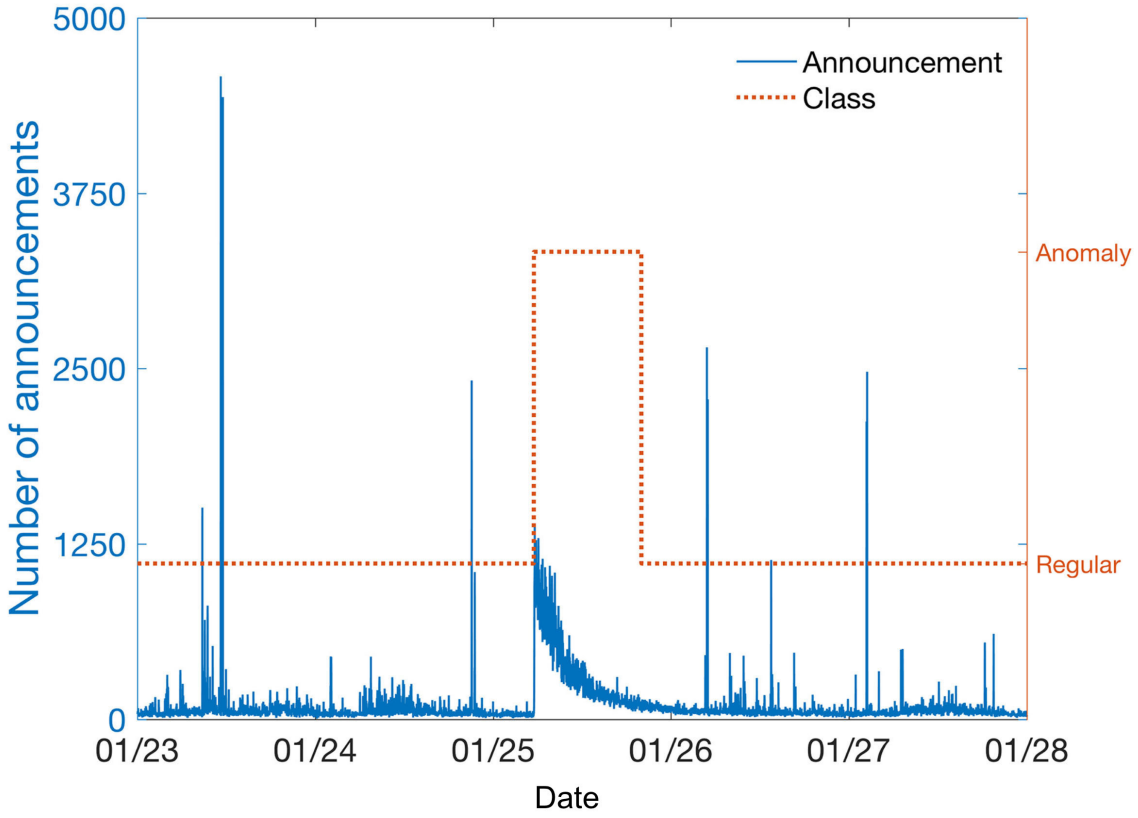

Figure 2.1: Number of BGP announcements occurred between January 23, 2003 and January 28, 2003. The announcements occurred during Slammer anomaly are labeled as the "anomaly" class while others belong to the "regular" class.

a small piece of code that generated IP addresses at random. Furthermore, code replicated itself by infecting new machines through randomly generated targets. If the destination IP address was a Microsoft SQL server or a user's PC with the Microsoft SQL Server Data Engine (MSDE) installed, the server became infected and began infecting other servers. The number of infected machines doubled approximately every nine seconds. As a result, the update messages consumed most of the routers' bandwidth, which in turn slowed down the routers and, in some cases, caused the routers to crash. Single infected machines have reported additional traffic of 50 Mbps [65] as a consequence of increased generation of update messages.

Nimda [63]: The Nimda worm [63] exploited vulnerabilities in the Microsoft Internet Information Services (IIS) web servers for the Internet Explorer 5 on September 18, 2001. It propagated fast through email messages, web browsers, and file systems. The worm propagated by sending an infected attachment that was automatically downloaded after viewing the email messages. A user could also download it from the website or access an

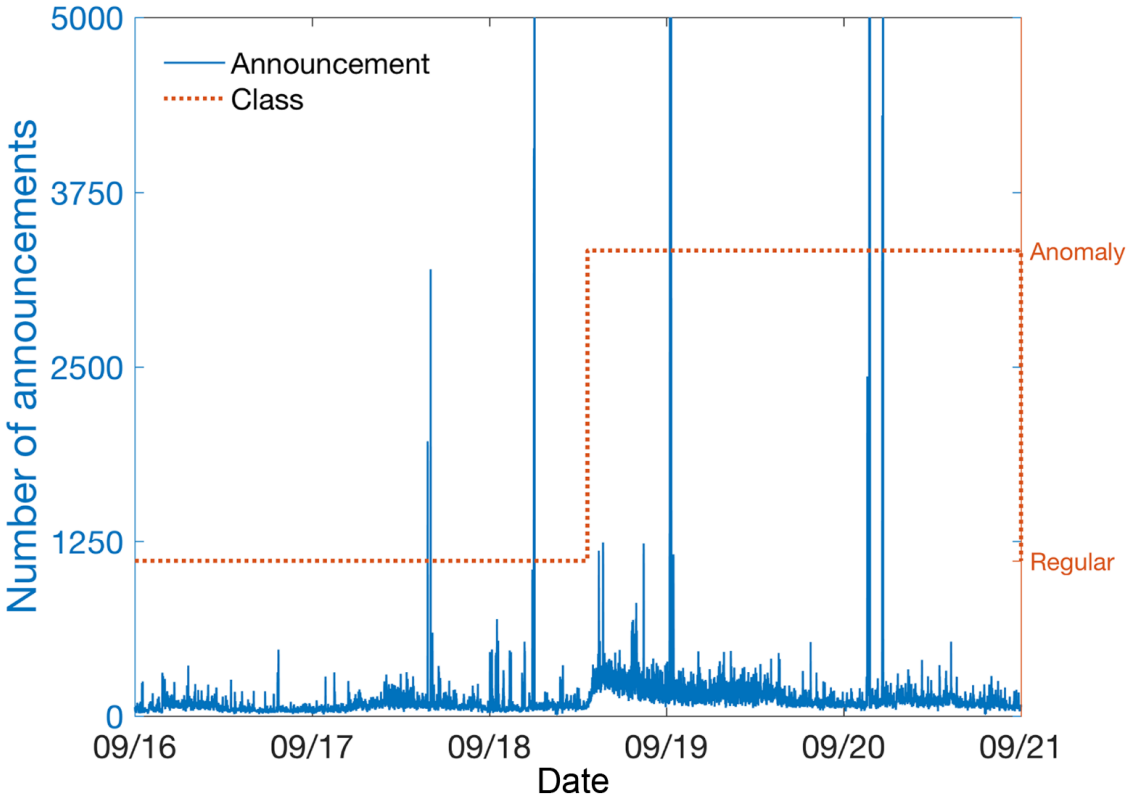

Figure 2.2: Number of BGP announcements occurred between September 16, 2001 and September 21, 2001. The number of announcements issued during Nimda anomaly are labeled as the "anomaly" class while others belong to the "regular" class.

infected file through the network. The worm modified the content of the web document file in the infected hosts and copied itself in local host directories.

Code Red I [64]: Although the Code Red I worm attacked Microsoft IIS web servers earlier, the peak of infected computers was observed on July 19, 2001. The worm affected approximately half a million IP addresses a day. It took advantage of vulnerability in the Internet Information Services (IIS) indexing software. The worm replicated itself by exploiting weakness of the IIS servers and, unlike the Slammer worm, Code Red I searched for vulnerable servers to infect. It triggered a buffer overflow in the infected hosts by writing to the buffers without checking their limits. Rate of infection was doubling every 37 minutes.

Panix domain hijack: Panix, the oldest commercial ISP in New York state, was hijacked on January 22, 2006. Its services were unreachable from the greater part of the Internet. Con Edison (AS 27506) advertised routes that it did not own at the time. Panix was previously a customer of Con Edison, which was once authorized to offer advertised routes. Even though AS 27506 originated improper routes, major downstream ISPs did not properly configure filters and propagated those routes, leading to excess number of update messages.

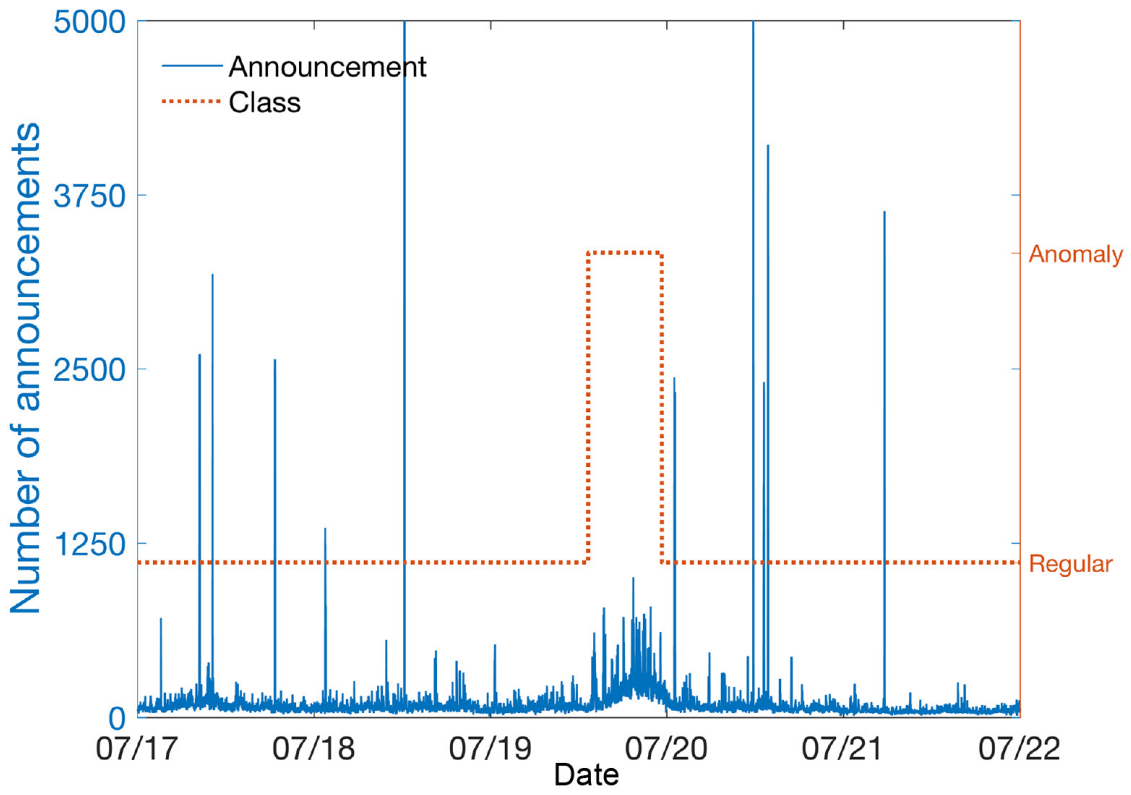

Figure 2.3: Number of BGP announcements issued from July 17, 2001 to July 22, 2001. The number of announcements occurred during Code Red I anomaly are labeled as the "anomaly" class while others belong to the "regular" class.

Table 2.2: Internet anomalous events.

Event	**Date**	**RRC**	**Peers**
Panix domain hijack	Jan. 2006	Route Views	AS 12956, AS 6762, AS 6939, AS 3549
AS 3356/AS 714 de-peering	Oct. 2005	RIS 01	AS 13237, AS 8342, AS 5511, AS16034
Moscow power blackout	May 2005	RIS 05	AS 1853, AS 12793, AS 13237
AS 9121 routing table leak	Dec. 2004	RIS 05	AS 1853, AS 12793, AS 13237
AS-path error	Oct. 2001	RIS 03	AS 3257, AS 3333, AS 6762, AS 9057
AS 3561 improper filtering	Apr. 2001	RIS 03	AS 3257, AS 3333, AS 286

De-Peering: The AS 3356/AS 714 De-Peering event occurred on October 5, 2005. Even though the Level 3 Communications (AS 3356) notified the Cogent Communications (AS 714) two months in advance of de-peering, the event created reachability problems for many Internet locations. Mostly affected were single-homed customers of Cogent (approximately

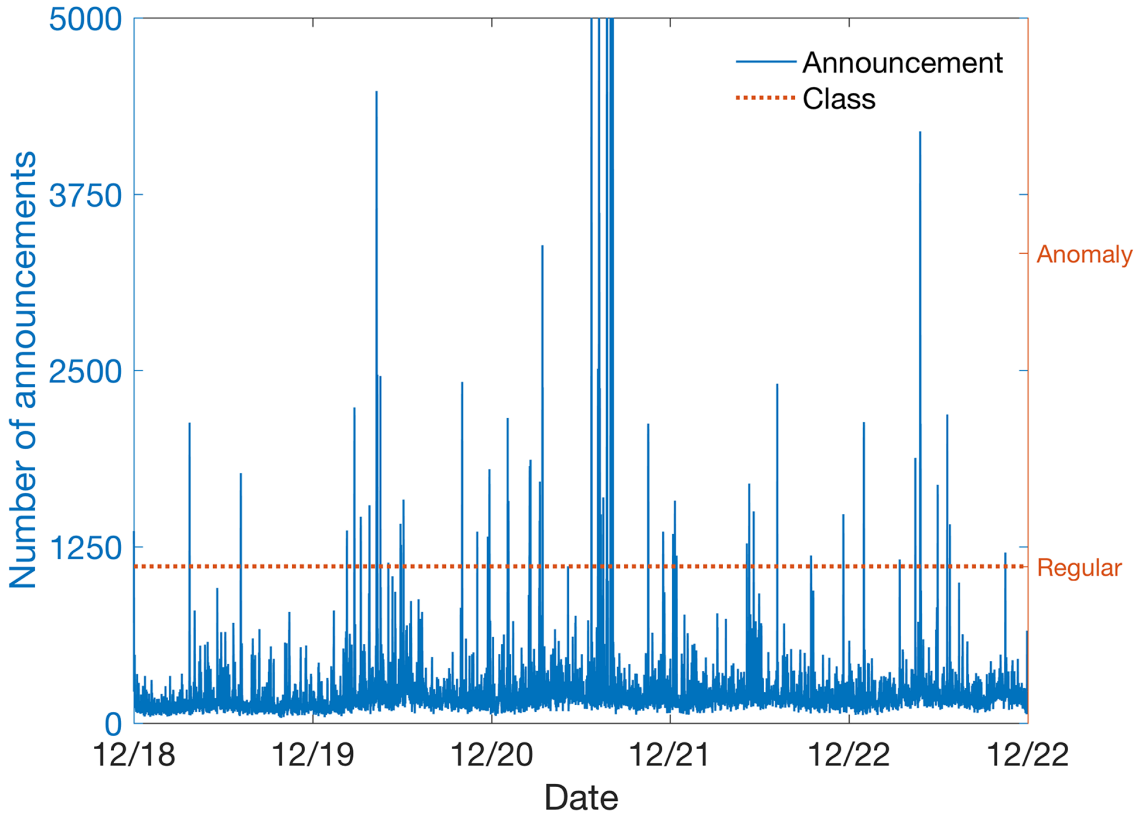

Figure 2.4: Number of BGP announcements occurred between December 18, 2001 and December 22, 2001. Shown is an example of BGP announcements occurred during regular traffic. The number of announcements belong to the "regular" class.

2,300 prefixes) and Level 3 Communications (approximately 5,000 prefixes). De-Peering resulted in partitioning of approximately 4 % of prefixes in the global routing table.

Moscow power blackout: The blackout occurred on May 25, 2005 and lasted several hours. The Moscow Internet exchange was shut down during the power outage. Routing instabilities were observed due to loss of connectivity of several ISPs peering at this exchange. This effect was apparent at the RIS remote route collector in Vienna (rrc05) through a surge in announcement messages arriving from peer AS 12793.

AS 9121 routing table leak: It occurred on December 24, 2004 when AS 9121 announced to peers that it could be used to reach almost 70% of all prefixes (over 106,000). As a consequence, numerous networks had either misdirected or lost their traffic. The AS 9121 began announcing prefixes to peers around 9:20 GMT and the event lasted until shortly after 10:00 GMT. It continued to announce bad prefixes throughout the day. The announcement rate reached the second peak at 19:47 GMT.

AS 3561 improper filtering: This was a BGP mis-configuration error that occurred on April 6, 2001. AS 3561 allowed improper route announcements from its downstream cus-

tomers, which created connectivity disruptions. Surge of announcement messages originating from peer AS 3257 was observed at the RIS rrc03.

AS-path error: The AS-path error occurred on October 7, 2001. It was caused by an abnormal AS-path (AS 3300, AS 64603, AS 2008) that contained private AS 64603 that should not have been included in the path. At the time, AS 3300 and AS 2008 belonged to INFONET Europe and INFONET USA, respectively. The path was distributed to the network via mis-configured routers and caused the leak of the private AS numbers.

2.2 Analyzed BGP Datasets

The Internet routing data used in this Thesis to detect BGP anomalies is acquired from projects that provide valuable information to networking research: the Route Views project [66] at the University of Oregon, USA and the Routing Information Service (RIS) project initiated in 2001 by the Réseaux IP Européens (RIPE) Network Coordination Centre (NCC) [67]. Both projects collect and store chronological routing data that offer a unique view of the Internet topology by establishing a BGP peering agreements with different ISP's around the world. The archives include recent data and hitorical data dating back to a decade. The Route Views and RIPE BGP update messages are publicly available to the research community. The regular BCNET dataset is collected at the BCNET location in Vancouver, British Columbia, Canada [68], [69]. We use RIPE BGP update messages originated from the AS 513 (route collector rrc04) member of the CERN Internet Exchange Point (CIXP). Only data collected during the periods of Internet anomalies are considered.

The Route Views project collects BGP routing tables from multiple geographically distributed BGP Cisco routers and Zebra servers every two hours. At the time of BGP anomalies considered in this study, two Cisco routers and two Zebra servers were located at the University of Oregon, USA. The remaining five Zebra servers are located at Equinix-USA, ISC-USA, KIXP-Kenya, LINX-Great Britain, and DIXIE-Japan [66]. Most participating ASes in the Route Views project are located in North America.

The RIPE NCC began collecting and storing Internet routing data in 2001 through the RIS project RIPE. The data were exported every fifteen minutes until July 2003. The interval between consecutive exports was later decreased to five minutes. BGP update messages are collected by the RRCs and stored in the multi-threaded routing toolkit (MRT) binary format [70]. The Internet Engineering Task Force (IETF) [71] introduced MRT to export routing protocol messages, state changes, and content of the routing information base (RIB). We converted BGP update messages from MRT into American Standard Code for Information Interchange (ASCII) format by using the *libBGPdump* library [72] on a Linux platform. *LibBGPdump* is a C library maintained by the RIPE NCC and it is used to analyze dump files, which are in MRT format.

BGPMon [73] is another tool for collecting BGP. It has been developed within the Oregon Route Views project. In addition to downloading data from the site, a user may also open a TCP connection and receive real-time data of both BGP update messages and routing tables. The advantage of using real-time data is that it decreases the delay caused by network propagation. The archived data may be delayed by several hours. BGPmon collects the data in the eXtensible Markup Language (XML) format that is self-descriptive. The format is similar to HyperText Markup Language (HTML). In the message, it includes both binary attributes and ASCII text. Thus, users could easily edit local tags and share the message. An example of the local tag is the timestamp that a BGP message should include. A Unix timestamp is preferred by MRT format while a human friendly listing is preferred by users. Some applications may require a finer granularity of milliseconds. The XML format includes all three types (ASCII, MRT, and combined) of time display. A user may select the suitable <TIME> tag in the message according to the requirement of applications.

Additional data sources of available data are such as Center for Applied Internet Data Analysis (Caida) [108], KDD Cup 1999 data [109], and Routing Assets Database (RADb) [110].

2.2.1 Processing of the Collected Data

BGP update messages are collected during the time period when the Internet experienced anomalies. Datasets are concatenated to increase the size of training datasets and, thus, improve the classification results. Anomaly datasets and their concatenations used for training and testing are shown in Table 2.3. Slammer, Nimda, and Code Red I anomalies are used to create three training and three test datasets. Each training dataset contains two anomalies with the corresponding test dataset contains the remaining anomaly data.

Table 2.3: Training and test datasets.

Training dataset	**Anomalies**	**Test dataset**
1	Slammer and Nimda	Code Red I
2	Slammer and Code Red I	Nimda
3	Nimda and Code Red I	Slammer

For Slammer and Code Red I anomalies, we consider a five-day period: the days of the attack (anomalous data points) and two days prior (regular data points) and two days after the attack (regular data points). Attacks of Slammer and Code Red I lasted for 869 and 600 minutes, respectively. The duration of regular data stream within two days before and after the Slammer and Code Red I are 6,331 and 6,600 minutes, respectively. The Nimda dataset is an exception because the attack lasted for two and half days (3,521 minutes). Hence, we only use two and half days prior to the event as regular data points (3,679 minutes). Each dataset consists of 14,400 ($2 \times 7,200$) data points represented by $14,400 \times 37$ matrix that corresponds to 37 features. In addition to anomalous test datasets, we also use regular datasets collected from RIPE [67] and BCNET [74].

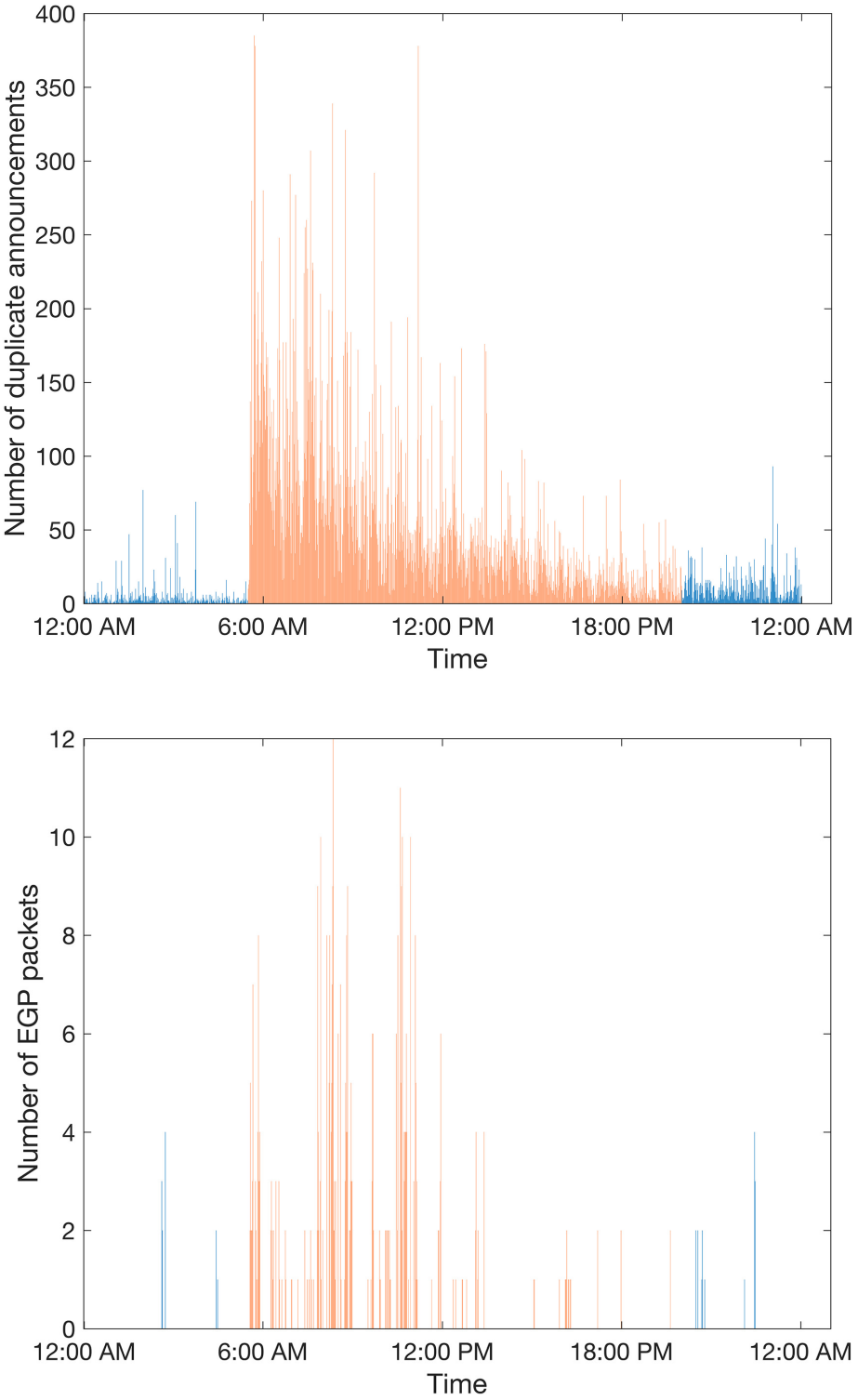

Figure 2.5: BGP announcements occurred during the Slammer worm attack: number of duplicate announcements (top) and number of EGP packets (bottom). The red streams (light grey) are anomalous data points and the blue (dark grey) ones are regular data points.

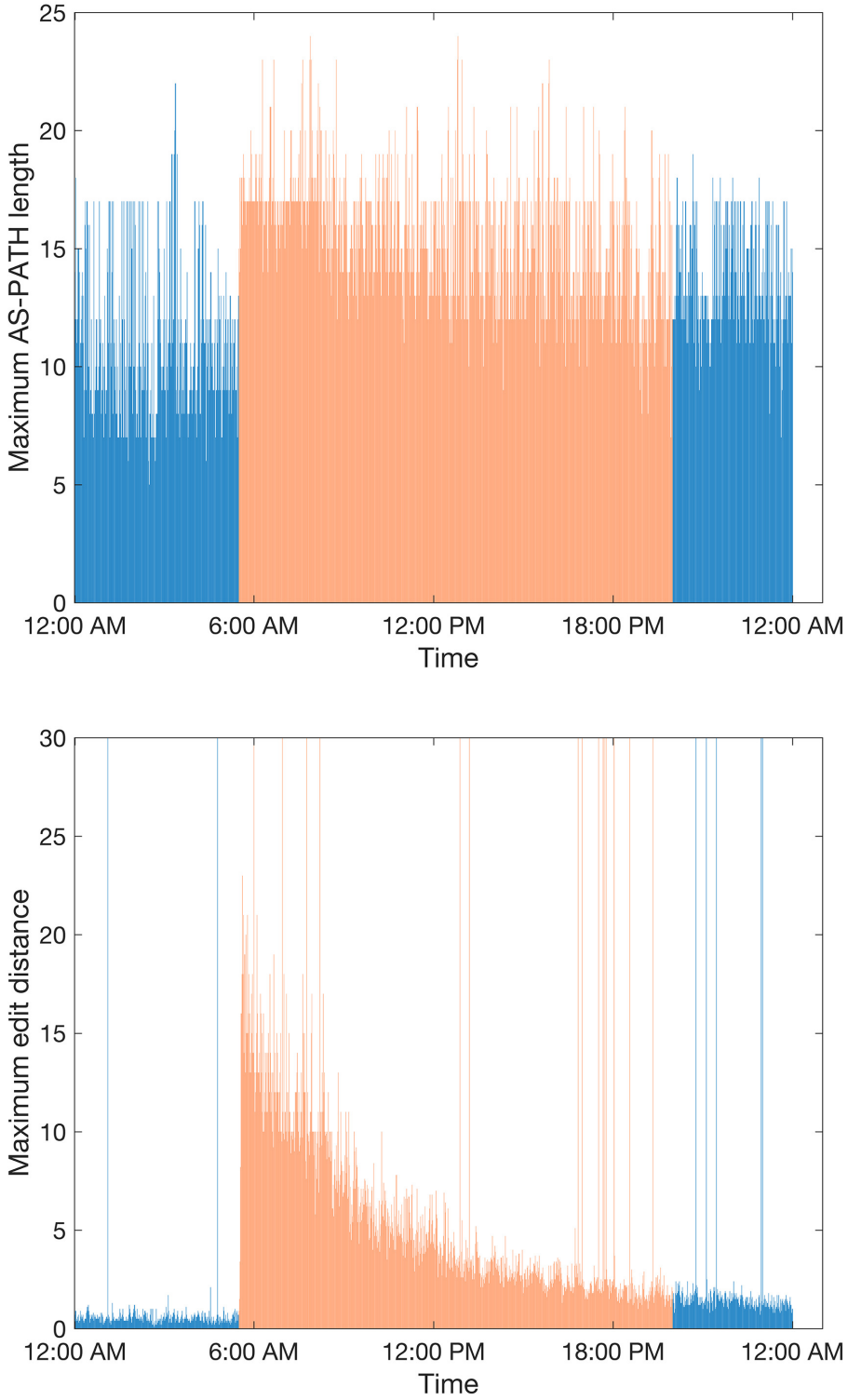

Figure 2.6: BGP announcements occurred during the Slammer worm attack: maximum AS-path length (top) and maximum AS-path edit distance (bottom). The red streams (light grey) are anomalous data points and the blue (dark grey) ones are regular data points.

Chapter 3

Extraction of Features from BGP Update Messages

Feature extraction is the first step in the classification process. We used a software tool (written in C#) to parse the ASCII files and extract statistics related to the desired features. The *AS-path* is a BGP update message attribute that enables the protocol to select the best path for routing packets. It indicates a path that a packet may traverse to reach its destination. If a feature is derived from the *AS-path* attribute, it is categorized as an *AS-path* feature. Otherwise, it is categorized as a *volume* feature. There are three types of features: continuous, categorical, and binary. Extracted *AS-path* and *volume* features are shown in Table 3.1 [75].

Definitions of the extracted features are listed in Table 3.2. BGP update messages are either announcement or withdrawal messages for the NLRI prefixes. The NLRI prefixes that have identical BGP attributes are encapsulated and sent in one BGP packet [76]. Hence, a BGP packet may contain more than one announced or withdrawn NLRI prefixes. The average and the maximum number of AS peers are used for calculating *AS-path* lengths. Duplicate announcements are the BGP update packets that have identical NLRI prefixes and the *AS-path* attributes. Implicit withdrawals are the BGP announcements with district *AS-paths* for already announced NLRI prefixes [77]. The edit distance is a metric to quantify the similarity of strings. A router uses edit distance to measure the difference between two AS paths. The edit distance between two *AS-path* attributes is the minimum number of deletions, insertions, or substitutions that need to be executed to match the two attributes [47]. For example, the edit distance between *AS-path* 513 940 and *AS-path* 513 4567 1318 is two because one insertion and one substitution are sufficient to match the two *AS-paths*. The more frequent changes in an AS path, the larger is the edit distance, which makes the routing update less trustworthy [78]. The maximum *AS-path* length and the maximum edit distance are used to count Features 14 to 33. We also consider Features 34, 35, and 36 based on distinct values of the origin attribute that specifies the origin of a BGP update packet and may assume three values: IGP, EGP, and incomplete. Even though

Table 3.1: List of features extracted from BGP update messages.

Feature	Name	Category
1	Number of announcements	*volume*
2	Number of withdrawals	*volume*
3	Number of announced NLRI prefixes	*volume*
4	Number of withdrawn NLRI prefixes	*volume*
5	Average *AS-path* length	*AS-path*
6	Maximum *AS-path* length	*AS-path*
7	Average unique *AS-path* length	*AS-path*
8	Number of duplicate announcements	*volume*
9	Number of duplicate withdrawals	*volume*
10	Number of implicit withdrawals	*volume*
11	Average edit distance	*AS-path*
12	Maximum edit distance	*AS-path*
13	Inter-arrival time	*volume*
14-24	Maximum edit distance = n, where $n = (7, ..., 17)$	*AS-path*
25-33	Maximum *AS-path* length = n, where $n = (7, ..., 15)$	*AS-path*
34	Number of Interior Gateway Protocol (IGP) packets	*volume*
35	Number of Exterior Gateway Protocol (EGP) packets	*volume*
36	Number of incomplete packets	*volume*
37	Packet size (B)	*volume*

the EGP protocol is the predecessor of BGP, EGP packets still appear in traffic traces containing BGP updates messages. Under a worm attack, BGP traces contained large volume of EGP packets. Furthermore, incomplete update messages imply that the announced NLRI prefixes are generated from unknown sources. They usually originate from BGP redistribution configurations [76]. Examples are shown in Table 3.3 while various distributions during the Slammer worm are shown in Fig. 3.1 and Fig. 3.2. During the Slammer worm attack, the number of autonomous systems included in the maximum *AS-path* length ranges from 6 to 24. The most frequent number of ASes in an AS-path is 12, which occurs more than 1,200 times. The edit distance reflects the changes in AS paths. During the Slammer worm, the paths change frequently.

Performance of the BGP protocol is based on trust among BGP peers because they assume that the interchanged announcements are accurate and reliable. This trust relationship is vulnerable during BGP anomalies. For example, during BGP hijacks, a BGP peer may announce unauthorized prefixes that indicate to other peers that it is the originating peer. These false announcements propagate across the Internet to other BGP peers and, hence, affect the number of BGP announcements (updates and withdrawals) worldwide.

The top selected *AS-path* features appear on the boundaries of the distributions. This indicates that during BGP anomalies, the edit distance and *AS-path* length of the BGP

Table 3.2: Definition of *volume* and *AS-path* features extracted from BGP update messages.

Feature	**Name**	**Definition**
1	Number of announcements	Routes available for delivery of data
2	Number of withdrawals	Routes no longer reachable
3/4	Number of announced/withdrawn NLRI prefixes	BGP update messages that have type field set to announcement/withdrawal
5/6/7	Average/maximum/average unique *AS-path* length	Various *AS-path* lengths
8/9	Number of duplicate announcements/withdrawals	Duplicate BGP update messages with type field set to announcement/withdrawal
10	Number of implicit withdrawals	BGP update messages with type field set to announcement and different *AS-path* attribute for already announced NLRI prefixes
11/12	Average/maximum edit distance	Average/maximum of edit distances of messages
34/35/36	Number of IGP, EGP or, incomplete packets	BGP update messages generated by IGP, EGP, or unknown sources

Table 3.3: Example of BGP features.

Time	**Definition**	**BGP update type**	**NLRI**	**AS-path**
t_0	Announcement	Announcement	199.60.12.130	13455 614
t_1	Withdrawal	Withdrawal	199.60.12.130	13455 614
t_2	Duplicate announcement	Announcement	199.60.12.130	13455 614
t_3	Implicit withdrawal	Announcement	199.60.12.130	16180 614
t_4	Duplicate withdrawal	Withdrawal	199.60.12.130	13455 614

announcements tend to have a very high or a very low value and, hence, large variance. This implies that during an anomaly attack, *AS-path* features are the distribution outliers. For example, approximately 58% of the *AS-path* features are larger than the distribution mean. Large length of the *AS-path* BGP attribute implies that the packet is routed to its destination via a longer path, which causes large routing delays during BGP anomalies. Similarly, very short lengths of *AS-path* attributes occur during BGP hijacks [6] when the new (false) originator usually gains a preferred or shorter path to the destination.

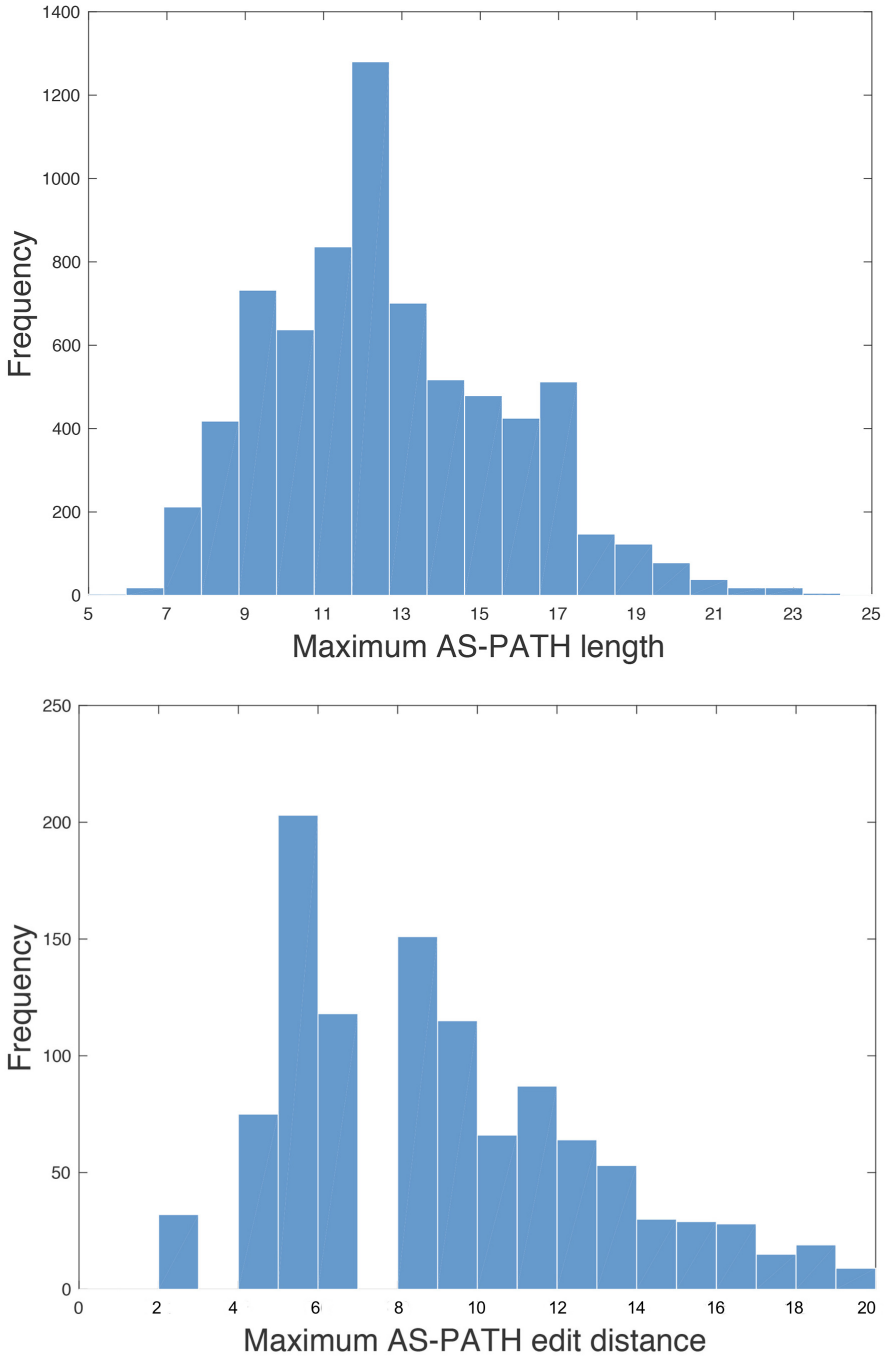

Figure 3.1: Distribution of the maximum *AS-path* length (top) and the maximum edit distance (bottom) collected during the Slammer worm. Shown maximum AS-paths contains up to 24 ASes, and the paths change frequently.

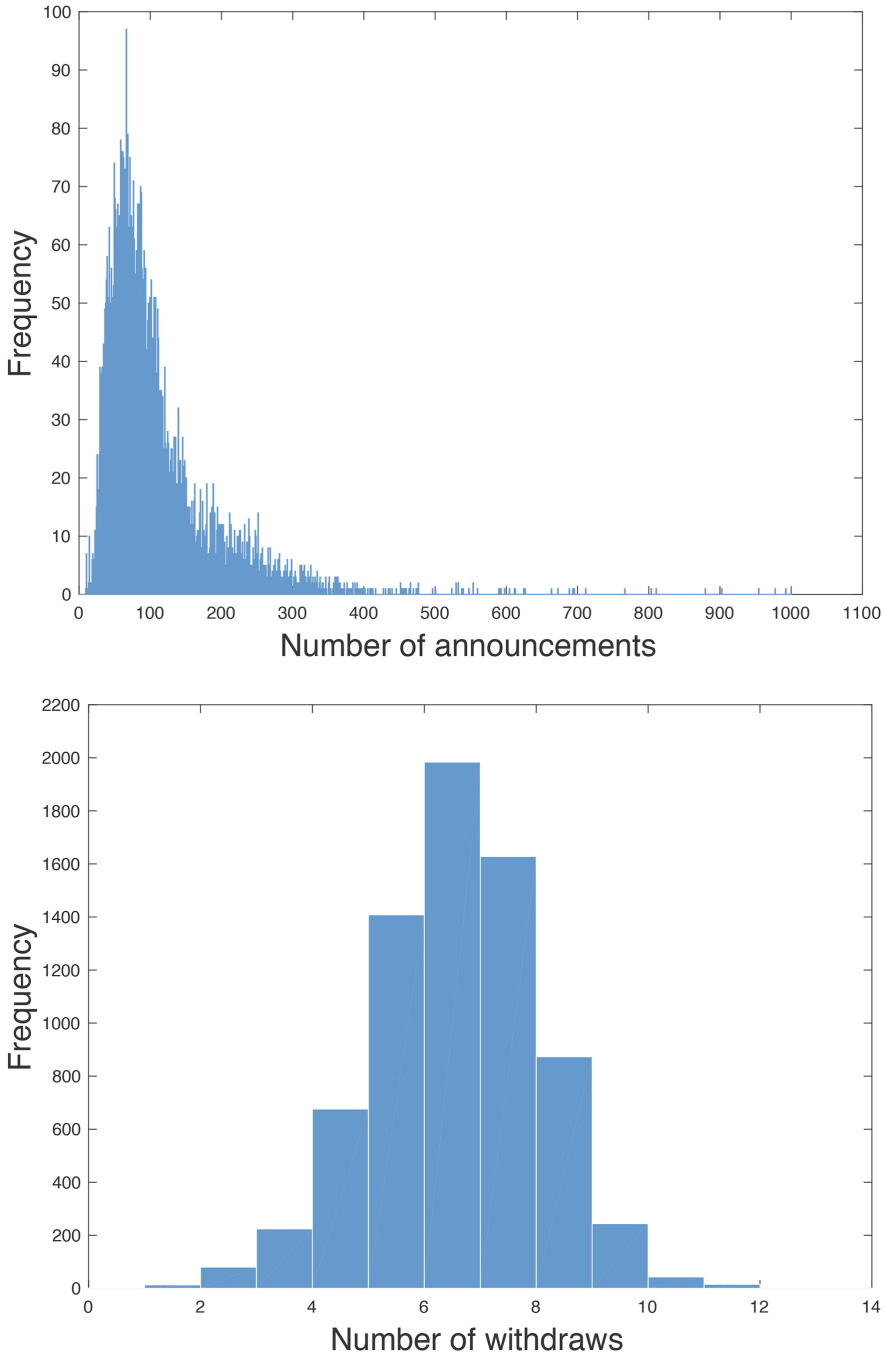

Figure 3.2: Distribution of the number of BGP announcements (top) and withdrawals (bottom) for the Code Red I worm.

Chapter 4

Performance Metrics and the Long Short-Term Memory Neural Network

Classification aims to identify various classes in a dataset. Each category in the classification domain is called a class. A classifier labels the data points as either anomaly or regular events. We consider datasets of known network anomalies and test the classifier's ability to reliably identify the anomaly class. Training and test datasets usually contain fewer anomalous samples compared to the regular data points. Classifier models are usually trained using datasets containing limited number of anomalies and are then applied on a test dataset. Performance of a classification model depends on a model's ability to correctly predict classes. Classifiers are evaluated based on various metrics such as accuracy, F-Score, precision, and sensitivity.

4.1 Introduction of Classification Algorithm

Most classification algorithms minimize the number of incorrectly predicted class labels while ignoring the difference between types of misclassified labels by assuming that all misclassifications have equal costs. The assumption that all misclassification types are equally costly is inaccurate in many application domains. In the case of BGP anomaly detection, incorrectly classifying an anomalous sample may be more costly than incorrect classification of a regular sample. As a result, a classifier that is trained using an unbalanced dataset may successfully classify the majority class with a good accuracy while being unable to accurately classify the minority class. A dataset is unbalanced when at least one class is represented by a smaller number of training samples compared to other classes. The Slammer and Code Red I anomaly datasets that have been used in this Thesis are unbalanced. In our studies, out of 7,200 samples, Slammer and Code Red I contain 869 and 600 anomalous events, respectively. The majority of samples are regular data. Only the Nimda dataset containing 3,521 anomalous events is more balanced compared to Slammer and Code Red I.

Various approaches have been proposed to achieve accurate classification results when dealing with unbalanced datasets. Examples include assigning a weight to each class or learning from one class (recognition-based) rather than two classes (discrimination-based) [79]. The weighted SVMs [29] assign distinct weights to data samples so that the training algorithm learns the decision surface according to the relative importance of data points in the training dataset. The fuzzy SVM [80], a version of weighted SVM, applies a fuzzy membership to each input sample and reformulates the SVM so that input points contribute differently to the learning decision surface. In this Thesis, we create the balanced datasets by randomly reducing the number of regular data points. Each balanced dataset contains the same number of regular and anomalous data samples.

4.2 Performance Metrics

The confusion matrix shown in Table 4.1 is used to evaluate performance of classification algorithms. True positive (TP) and False negative (FN) are the number of anomalous data points that are classified as anomaly and regular, respectively. False positive (FP) and True negative (TN) are the number of regular training data points that are classified as anomaly and regular, respectively.

Table 4.1: Confusion matrix.

	Predicted class	
Actual class	Anomaly (positive)	Regular (negative)
Anomaly (positive)	TP	FN
Regular (negative)	FP	TN

Variety of performance measures are calculated to evaluate classification algorithms, such as accuracy and F-Score:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$\text{F-Score} = 2 \times \frac{\text{precision} \times \text{sensitivity}}{\text{precision} + \text{sensitivity}}, \tag{4.2}$$

where

$$\text{precision} = \frac{TP}{TP + FP} \tag{4.3}$$

$$\text{sensitivity (recall)} = \frac{TP}{TP + FN}. \tag{4.4}$$

As a performance measure, accuracy reflects the true prediction over the entire dataset. It is commonly used in evaluating the classification performance. Accuracy assumes equal cost for misclassification and relatively uniform distributions of classes. It treats the regular

data points to be as important as the anomalous points. Hence, it may be an inadequate measure when comparing performance of classifiers [81] and misleading in the case of unbalanced datasets. The F-Score, which considers the false predictions, is important for anomaly detection because it is a harmonic mean of the precision and sensitivity, which measure the discriminating ability of the classifier to identify classified and misclassified anomalies. Precision identifies true anomalies among all data points that are correctly classified as anomalies. Sensitivity measures the ability of the model to identify correctly predicted anomalies.

As an example, consider a dataset that contains 900 regular and 100 anomalous data points. If a classifier identifies these 1,000 data points as regular, its accuracy is 90%, which seems high at the first glance. However, no anomalous data point is correctly classified and, hence, the F-Score is zero. Hence, the F-Score is often used to compare performance of classification models. It reflects the success of detecting anomalies rather than detecting either anomalies or regular data points. In this Thesis, we use both accuracy and F-Score to compare various classification algorithms.

4.3 Long Short-Term Memory (LSTM) Neural Network

A typical recurrent neural network (RNN) has a short-term memory because they use feedback connections to store recent input instances in the form of activations. However, the weights in RNNs have long-term memory because they change slowly during the training phase [28]. Traditional RNNs are designed to store inputs in order to predict the outputs [82]. However, they perform poorly when they need to bridge segments of information that have long-time gaps. As an alternative to traditional RNN architecture, LSTM is an intermediate model that enables short-term memory to last for a long period of time. The activations of the LSTM network have short-term memory while the weights correspond to long-term memory. Unlike traditional RNNs, LSTM networks are capable of concatenating time intervals to form a continuous memory [28]. LSTM, in conjunction with an appropriate gradient-based learning algorithm, was introduced to overcome long-term dependency and vanishing gradient problems [83].

4.3.1 Vanishing Gradient Problem

RNNs have simple structures and are effective for classification tasks. However, they are hard to train in practice due to the vanishing gradient problem, where the gradient becomes vanishingly small. The network may stop updating the values of weights and, thus, may stop the neural network from further training.

A simple example is a RNN with an input node, a hidden layer, and an output node as shown in Fig. 4.1. The hidden layer is associated with a weight, and the output is a function of its weight and the input:

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x}). \tag{4.5}$$

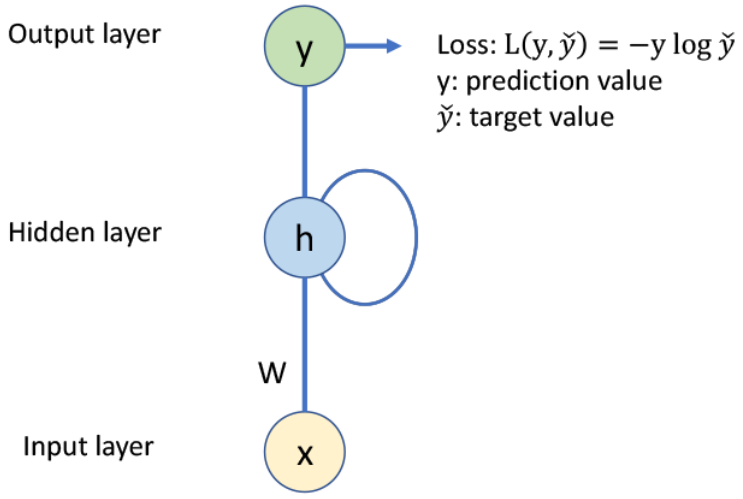

Figure 4.1: Shown is an example of a simple RNN with an input layer, a hidden layer, and an output layer. The loss is calculated using the prediction value and the target value.

At the end of the training, we calculate the difference between the prediction value and the actual target value named loss (error). We denote the actual target vector as $\hat{\boldsymbol{y}}$. The cross entropy loss is calculated as:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\boldsymbol{y} \log \hat{\boldsymbol{y}}. \tag{4.6}$$

The network then employs backpropagation in order to adjust weights and minimize the error. First, the chain rule is applied to calculate the gradient of the loss with respect to $\boldsymbol{W}$:

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{L}}{\partial f(\boldsymbol{y})} \cdot \frac{\partial f(\boldsymbol{y})}{\partial \boldsymbol{L}} = \frac{\partial \boldsymbol{L}}{\partial f(\boldsymbol{y})} \cdot f'(\boldsymbol{y}) \cdot \boldsymbol{W}. \tag{4.7}$$

In a typical RNN, weights are usually initialized based on the Gaussian distribution with mean zero and standard deviation one, which implies that $\boldsymbol{W} < 1$ in most cases. The activation function $f(\boldsymbol{y})$ is usually an S-shaped sigmoid function with returned value [0,1] defined as:

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-x}}, \tag{4.8}$$

where $\boldsymbol{x}$ is the input vector. The derivative of sigmoid function is always smaller than 0.25. Thus, the result of the gradient will be rather small and may even "vanish." In the opposite scenario, if the weight happens to be a large value, then the gradient will "explode."

The selection of activation functions is critical to prevent vanishing gradient. For example, in the case of the rectified linear unit (ReLU) function is defined as:

$$\begin{aligned} &R(\boldsymbol{x}) = max(0, x), \\ &R(\boldsymbol{x}) = 0 \quad when \quad x < 0, \\ &R(\boldsymbol{x}) = x \quad when \quad x \geqslant 0. \end{aligned} \tag{4.9}$$

The derivative of ReLU function is one when $x > 0$. Therefore, ReLU creates a more stable model compared to Sigmoid function.

4.3.2 LSTM Module

The LSTM module that is considered in this Thesis consists of an input layer, a single hidden LSTM layer, and an output layer. The input layer consists of 37 nodes (each node corresponds to a feature) that serve as inputs to the LSTM layer, which consists of self-connected LSTM cells called the "memory blocks" [84]. An LSTM module may contain multiple LSTM cells. The nth LSTM cell is composed of an input node x_t, a multiplicative input gate i_{nt}, internal state c_t, a forget gate f_{nt}, and a multiplicative output gate f_{nt}. An LSTM module is shown in Fig. 4.2 [85]. Components of an LSTM cell are described as follows [28], [84]:

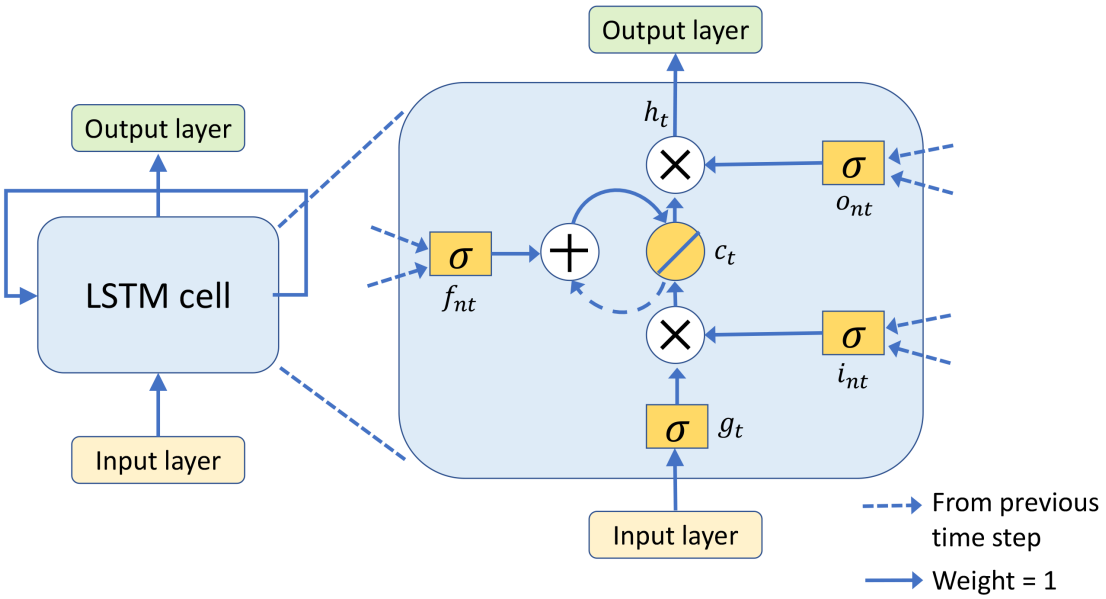

Figure 4.2: Repeating module for the LSTM neural network. Shown are the input layer, LSTM layer with one LSTM cell, and output layer.

- *Input node* g_t contains information of current input instances and cell states from other LSTM cells in the network. This information is then combined and passed through the input node with a *tanh* activation function. Thus, the generated value of the input node is [-1,1].

- *Input gate* i_{nt} connects the input node with the internal state. Inputs to the input gate and to the input node are independent. Similarly with traditional RNNs, inputs

include the current input data points and hidden states from the previous time step. The input gate controls the information that will be updated in the LSTM cell by multiplying the generated value of the input node. The input gate applies a linear function to the input data, followed by a logistic activation function (typically a *sigmoid* function) applied to output a value [0,1]. The information flows to the internal state if the value is 1.

- *Internal state* c_t is placed at the heart of each memory cell. It has an activation function such as the tanh function and a self-connected recurrent loop. The loop is modulated by the forget gate. The weight of the loop is [0,1], which is equal to the value of the forget gate. The internal state is the key to solve vanishing gradient problems.

- *Forget gate* f_{nt} determines whether to remember or discard the memories of the internal state [86]. Similar as the input gate, the forget gate applies a linear function to its inputs (including the previous hidden state and the new input) and uses a logistic activation function to generate value 1 to keep the memory or 0 to discard the information.

- *Output gate* o_{nt} works as a filter to clear irrelevant memories that are currently stored in the cell and, thus, controls the output. It has the same form as the input and forget gates and generates a value [0,1]. Whether the information may pass to the rest of the network is determined by the value of the internal state that multiplies the value generated by the output gate. Typically, the internal state runs a *tanh* activation function and outputs values [-1,1]. We used $ReLU$ activation function in this Thesis because $ReLU$ has a greater dynamic range and, thus, it is easier to train and to converge. If the output value is 1, the information may pass through the network.

At time t, the input node g_t, input gate i_{nt}, forget gate f_{nt}, and output gate o_{nt} in LSTM cell n are calculated as:

$$\boldsymbol{g_{nt}} = \tanh(\boldsymbol{U_{g_n} h_{t-1}} + \boldsymbol{W_g x_t} + \boldsymbol{b_{g_n}}) \tag{4.10}$$

$$\boldsymbol{i_{nt}} = \sigma(\boldsymbol{U_{i_n} h_{t-1}} + \boldsymbol{W_i x_t} + \boldsymbol{b_{i_n}}) \tag{4.11}$$

$$\boldsymbol{f_{nt}} = \sigma(\boldsymbol{U_{f_n} h_{t-1}} + \boldsymbol{W_f x_t} + \boldsymbol{b_{f_n}}) \tag{4.12}$$

$$\boldsymbol{o_{nt}} = \sigma(\boldsymbol{U_{o_n} h_{t-1}} + \boldsymbol{W_o x_t} + \boldsymbol{b_{o_n}}), \tag{4.13}$$

where the *sigmoid* activation function is denoted by σ and tangent function is denoted by *tanh*. Parameters U_{*_n} and W_* represent the weights. The output from the hidden layer at the previous time step is denoted as h_{t-1} while the input at the current time step is denoted as x_t. Biases are denoted by b_{*_n}. An activation function outputs a value 0 or 1 for each gate. The value determines how much information a gate may remain. For example, value 1 implies that the gate keeps all information while 0 implies that it cleans the entire memory.

The LSTM cell updates the internal state at each time step. At time t, the cell state c_t is calculated as:

$$\boldsymbol{c_t} = \boldsymbol{f_{nt}} * \boldsymbol{c_{t-1}} + \boldsymbol{i_{nt}} * \tanh(\boldsymbol{U_c h_{t-1}} + \boldsymbol{W_c x_t} + \boldsymbol{b_c}). \tag{4.14}$$

The *tanh* activation function is used to return a value in the range [-1, 1]. The output of an LSTM cell is calculated as:

$$\boldsymbol{h_t} = \boldsymbol{o_{nt}} * ReLU(\boldsymbol{c_t}), \tag{4.15}$$

where vector notation refers to values of nodes in an entire layer of cells. For example, $\boldsymbol{f}$ is a vector of the value of the forget gate at each LSTM cell in a layer while $\boldsymbol{f_n}$ is used to index the individual LSTM cell n.

The internal state is the key to help LSTM networks prevent the vanishing gradient problem. The actions of the internal state based on values of input and forget gates are shown in Table 4.2. The weight of the internal state is essentially the output of the forget gate, and the LSTM algorithm sets the value of the forget gate to one for important information within the internal state. Thus, if the forget and output gates are on and the input gates is off, the gradients of the internal state is passed through the network unchanged at each time step. Consequently, the LSTM architecture helps avoid the vanishing gradients problem.

Table 4.2: The actions of the internal state corresponding to values of input and forget gates.

Input gate	**Forget gate**	**Actions of the internal state**
0	1	Keep the memory from the previous time step
1	1	Add the current information to the memory
0	0	Discard both current and the past information
1	0	Overwrite the memory by the current information

Keras [87] is an open source neural networks Application Program Interface (API) written in Python. It is capable of running on top of TensorFlow [88], Deeplearning4j [89], Microsoft Cognitive Toolkit [90], or Theano [91]. Keras focuses on enabling fast experimentation with deep neural networks. It was developed as a part of the research project Open-ended Neuro-Electronic Intelligent Robot Operating System (ONEIROS) [92]. Keras is an interface that makes it easy to configure neural networks regardless of the backend computing library. In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Keras became TensorFlow's default API [92]. It is the first high-level library added to core TensorFlow.

Keras provides a user-friendly environment. Users may use Keras without interacting with the underlying backend engine. It is also an object-oriented library where all components including layers, cost functions, optimizers, activation functions, and regularization

schemes are individual models. For example, 'model.layers[3].output' is the output tensor of the 3rd layer in the model. These fully-configured models may be combined together or added with new models with little restrictions.

We use Keras with Python 2.7 to generate LSTM models. As the first step, we import the sequential model from Keras. The generated LSTM sequential models contain 37-dimensional inputs, 1 hidden layer, and 1-dimensional outputs. The original BGP dataset is split into training (70%) and test (30%) datasets. Biased weights are assigned to features having large variations. Thus, we normalize the data points and scale their values within the range [0, 1] in order to set the same importance to all features. In the original dataset, anomaly samples are labeled as -1. We replace these labels by 0 for the sake of computation because Keras does not recognize -1 as a label. We implement the *look_back* function to calculate the number of previous time steps that are used as input variables to predict the data point at next time step. The length of time sequence is set to 20. We empirically set the *look_back* value to 19, which implies each prediction is based on 19 previous and 1 current instances. The architecture of implemented LSTM classifier is shown as Figure 4.3.

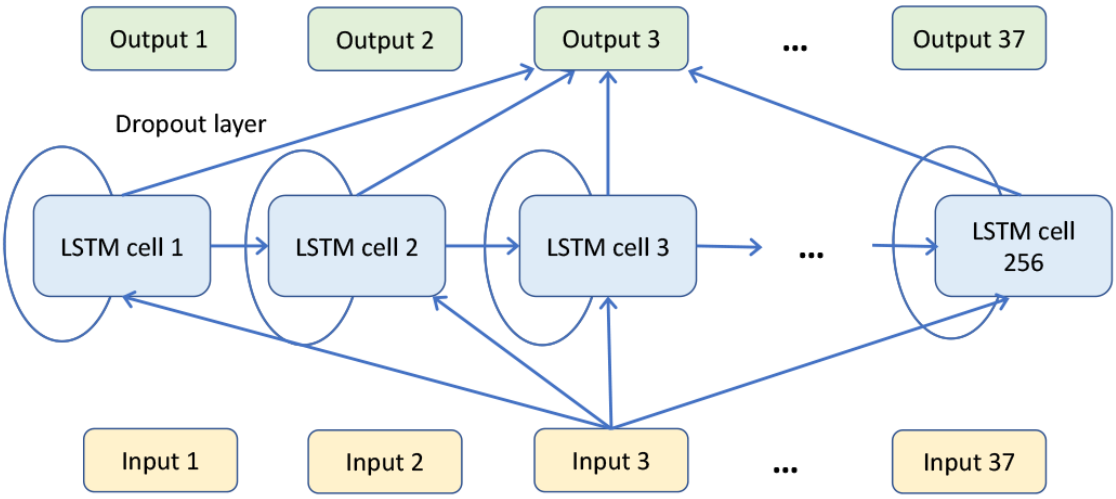

Figure 4.3: Architecture of the employed LSTM classifier. Shown are input layer with 37 nodes, LSTM layer with 256 cells, the dropout layer with 50% dropout rate, and the output layer.

The implemented LSTM model contains an input layer with 37-dimensional input nodes, a hidden layer with 256 LSTM cells, a dropout layer with 50% dropout rate, and an output layer that produces one result at each time step. The dropout [93] layer lies between the hidden layer and the output layer. Its function is to randomly discard LSTM blocks and their connections from the neural network during training. It effectively prevents overfitting for large neural networks. The LSTM layer uses the $ReLU$ activation function while the output

layer uses the *sigmoid* function. The "Adam" optimizer [94] offers superior performance when dealing with large datasets and high-dimensional parameter spaces. Thus, we use the "Adam" optimizer with the learning rate "lr = 0.001" to calculate the gradients and update the weight when compiling LSTM models. After the model is created, a random seed is used to initialize a pseudorandom number generator. We set the value of random seed to 77 in order to ensure the results are reproducible (otherwise, the system generates a random seed number at each time that would lead to a different result). The batch size needs to be defined when using Keras library. It increases the efficiency of the algorithm by setting the number of samples to be trained in the network during the learning phase. We set the batch size to 32, which implies 32 instances are trained together and 32 predictions are made each time. We use 20% of the original training dataset to validate the model. The parameters that lead to a desirable classification result are saved and are then applied in the test phase. To avoid overfitting, 30 trials are made for all tested datasets.

Chapter 5

Description of Classification Algorithms used for Comparison

We describe here machine learning classification techniques used to detect BGP anomalies that are reported in our previous studies. We then compare their results with the LSTM classifier in terms of classification accuracy and F-Score in order to examine the effectiveness of various BGP anomaly classifiers.

5.1 Support Vector Machine (SVM)

SVM was introduced as a technique for pattern recognition, the main author is Vapnik [95]. It is a supervised learning algorithm used for classification and regression tasks. It has been successfully applied to various areas such as face recognition [96], text-categorization [97], and BGP anomaly detection [12]. It outperforms neural networks in various applications.

In this Thesis, SVM is applied as a binary classifier for a classification task rather than regression. Given a set of labeled training samples, the SVM algorithm learns a classification hyperplane (decision boundary) by maximizing the minimum distance between data points belonging to various classes. There are two types of SVM models: hard-margin and soft-margin [95]. The hard-margin SVMs require that each data point is correctly classified while the soft-margin SVMs allow some data points to be misclassified. Soft-margin SVM allows flexible margin, which implies assigning small weights to data points on the incorrect side of the margin to reduce their influence. Illustration of the soft margin is shown in Fig. 5.1 [8]. The solid line indicates the decision boundary while dashed lines indicate the margins. Encircled data points are support vectors. The maximum margin is the perpendicular distance between the decision boundary and the closest support vectors. The SVM classifier finds the only hyperplane that maximizes the margin between two data sets. Data points for which $\zeta = 0$ are correctly classified and are either on the margin or on the correct side of the margin. Data points for which $0 \leq \zeta < 1$ are also correctly classified because they lie inside the margin and are on the correct side of the decision boundary. Data points

for which $\zeta > 1$ lie on the wrong side of the decision boundary and are misclassified. The outputs 1 and -1 correspond to anomaly and regular data points, respectively. The SVM solution maximizes the margin between the data points and the decision boundary. Data points that have the minimum distance to the decision boundary are called support vectors.

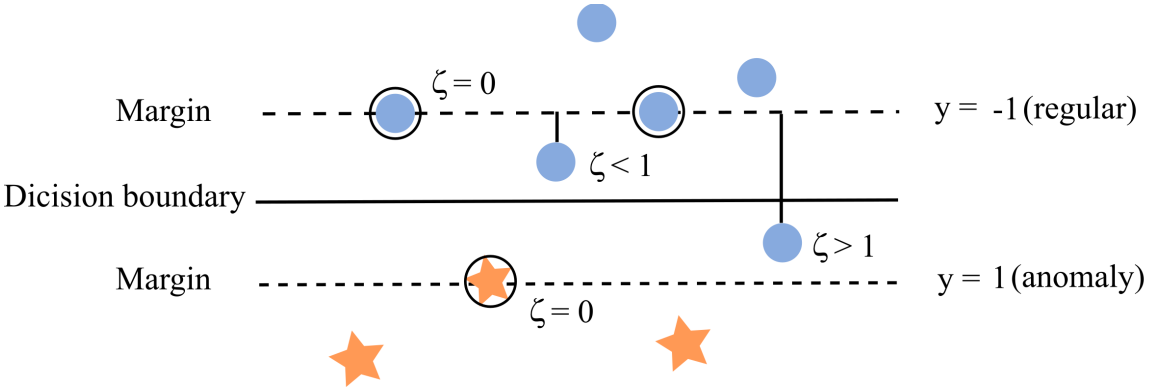

Figure 5.1: Illustration of the soft margin SVM [8]. Shown are correctly and incorrectly classified data points. Regular and anomalous data points are denoted by circles and stars, respectively. The circled points are support vectors.

The objective of SVM is to find the optimal hyperplane (a generalization of a plane): a hyperplane is a point in one dimension; a line in two dimentsions; a plane in three dimensions; a hyperplane in more than three dimensions. Multiple valid hyperplanes may be generated that successfully separate input data sets in a feature space. However, only optimal hyperplane ensures that maximum margin is achieved between both classes. The hyperplane is acquired by minimizing the loss function [8]:

$$C \times \sum_{n=1}^{N} \zeta_n + \frac{1}{2}||w||^2,$$
$$\text{with constraints: } t_n y(x_n) \geqslant 1 - \zeta_n, \ n = 1, ..., N, \tag{5.1}$$

where the regularization parameter C controls the trade-off between the slack variable ζ_n, N is the number of data points, and $\frac{1}{2}||w||^2$ is the margin. The regularization parameter $C > 0$ is used to avoid over-fitting problem. The target value is denoted by t_n while $y(x_n)$ and x_n are the training model and data points, respectively. The SVM solves a loss function as an optimization problem (5.1).

Instead of employing a minimization model (5.9), the problem may be formulated using Lagrangian dual multiplier β as:

$$\max \sum_{n=1}^{N} \beta_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \beta_n \beta_m y_n y_m (\boldsymbol{x}_n \cdot \boldsymbol{x}_m), \tag{5.2}$$

subject to:

$$0 \leq \beta_i \leq C \ \ \forall \ \ i = 1, 2, ..., n \sum_{i=1}^{n} \beta_i y_i = 0. \tag{5.3}$$

Find the optimal hyperplane in high dimensional feature space is complicated and computationally expensive. Therefore, the SVM employs a nonlinear kernel function to map input space to a feature space where linear separation is possible [98]. Illustration of the role of a kernel function is shown as Fig. 5.2.

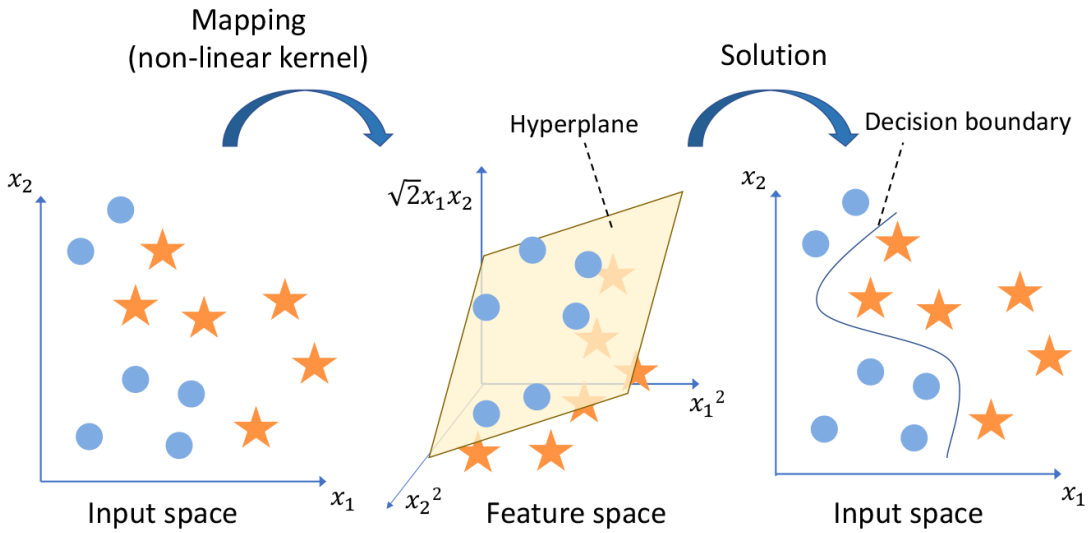

Figure 5.2: Illustration of the role of a kernel function. Blue circles and yellow stars are regular and anomalous data points, respectively. A nonlinear kernel function maps the input data points from input space to a higher dimensional feature space and calculates an optimal separating hyperplane. Then the hyperplane is mapped back to input space and results in a nonlinear decision boundary.

In the training phase, we first calculate scalar inner products $x_k \cdot x_l$ of the training data points. Their mapping from an input space to a feature space may be achieved by substituting the inner product:

$$\boldsymbol{x}_n \cdot \boldsymbol{x}_m \mapsto \phi(\boldsymbol{x}_n) \cdot \phi(\boldsymbol{x}_m), \tag{5.4}$$

where $\boldsymbol{x_k}$ and $\boldsymbol{x_l}$ are feature vectors. The mapping function is denoted as ϕ.

Instead of calculating each ϕ, a kernel function $k(x_k, x_l)$ is used. A kernel function takes two feature vectors as arguments and calculates the value of their inner product:

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m). \tag{5.5}$$

The result is returned to a new input space and generates a decision boundary for input data points. The advantage of using the "kernel trick" is that the complexity of the optimization

problem only depends on the input space instead of the feature space [98]. The objective function for nonlinear SVM has the form [99]:

$$\max \sum_{n=1}^{N} \beta_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \beta_n \beta_m y_n y_m k(\boldsymbol{x}_n, \boldsymbol{x}_m). \tag{5.6}$$

The Radial Basis Function (RBF) is often chosen [12] because it creates a large function space and outperforms other types of SVM kernels [8]. The RBF kernel k is used to avoid the high dimension of the feature matrix:

$$k(\boldsymbol{x_n}, \boldsymbol{x_m}) = exp(-\|\boldsymbol{x_n} - \boldsymbol{x_m}\|^2 / 2\sigma^2). \tag{5.7}$$

It relies on the Euclidean distance between $\boldsymbol{x_n}$ and $\boldsymbol{x_m}$ feature vectors. The datasets are trained using 10-fold cross validation to select parameters $(C, 1/2\sigma^2)$ that gives the best accuracy.

5.2 Naïve Bayes

The Naïve Bayes classifiers are among the most efficient machine learning classification techniques. The generative Bayesian models are used as classifiers using labeled datasets. They assume conditional independence among features, implying that given a class label, the probability distribution of each event is not influenced by the occurrence of other events. Hence,

$$\Pr(\mathbf{X}_k = \boldsymbol{x}_k, \mathbf{X}_l = \boldsymbol{x}_l | c_j) = \Pr(\mathbf{X}_k = \boldsymbol{x}_k | c_j) \Pr(\mathbf{X}_l = \boldsymbol{x}_l | c_j), \tag{5.8}$$

where $\boldsymbol{x}_k$ and $\boldsymbol{x}_l$ are realizations of feature vectors $\mathbf{X}_k$ and $\mathbf{X}_l$, respectively. In a two-way classification, classes labeled $c_1 = 1$ and $c_2 = -1$ denote anomalous and regular data points, respectively. An arbitrary training data point $\boldsymbol{x}_i$ is classified as anomalous if the posterior $\Pr(c_1 | \mathbf{X}_i = \boldsymbol{x}_i)$ is larger than $\Pr(c_2 | \mathbf{X}_i = \boldsymbol{x}_i)$. Even though it is naive to assume that features are independent for a given class (5.8), for certain applications Naïve Bayes classifiers perform better compared to other classifiers. They have low complexity, may be trained effectively with smaller datasets, and may be used for online real time detection of anomalies.

The probability distributions of the priors $\Pr(c_j)$ and the likelihoods $\Pr(\mathbf{X}_i = \boldsymbol{x}_i | c_j)$ are estimated using the training datasets. Posterior of a data point represented as a row vector $\boldsymbol{x}_i$ is calculated using the Bayes rule:

$$\begin{aligned} \Pr(c_j | \mathbf{X}_i = \boldsymbol{x}_i) &= \frac{\Pr(\mathbf{X}_i = \boldsymbol{x}_i | c_j) \Pr(c_j)}{\Pr(\mathbf{X}_i = \boldsymbol{x}_i)} \\ &\approx \Pr(\mathbf{X}_i = \boldsymbol{x}_i | c_j) \Pr(c_j). \end{aligned} \tag{5.9}$$

The naive assumption of independence among features helps calculate the likelihood of a data point as:

$$\Pr(\mathbf{X}_i = \boldsymbol{x}_i | c_j) = \prod_{k=1}^{K} \Pr(X_{ik} = x_{ik} | c_j), \tag{5.10}$$

where K denotes the number of features. The probabilities on the right-hand side (5.10) are calculated using the Gaussian distribution $\mathcal{N}$:

$$\Pr(\mathbf{X}_{ik} = \boldsymbol{x}_{ik} | c_j, \mu_k, \sigma_k) = \mathcal{N}(X_{ik} = x_{ik} | c_j, \mu_k, \sigma_k), \tag{5.11}$$

where μ_k and σ_k are the mean and standard deviation of the k^{th} feature, respectively. We assume that priors are equal to the relative frequencies of the training data points for each class c_j. Hence,

$$\Pr(c_j) = \frac{N_j}{N}, \tag{5.12}$$

where N_j is the number of training data points that belong to the j^{th} class and N is the total number of training data points.

5.3 Decision Tree Algorithm

The Decision Tree approach is commonly used in data mining to predict the class labels based on several input variables. A classification tree is a directed tree where the root is the source dataset and each internal (non-leaf) node is labeled with an input feature. The tree branches are prediction outcomes that are labeled with possible feature values while each leaf node is labeled with a class or a class probability distribution [100]. The advantage of using Decision Tree is that the feature selection is not required for data preprocessing because Decision Tree algorithm automatically selects the important features during the classification. Moreover, Decision Tree does not require linear datasets. A top-down approach is commonly used for constructing Decision Trees, which implies that the tree is divided from a root node into subsets that contain homogeneous data points.

The Decision Tree algorithm is one of the most successful supervised learning techniques [31]. During the Decision Tree learning, each input instance is represented by features, which are categorical or continuous variables. A tree is "learned" by splitting the input dataset into subsets based on appropriate features. This process is repeated on each derived subset using recursive partitioning until the splitting no longer adds values to the predictions. After a Decision Tree is learned, each path from the root node (source) to a leaf node is transformed into a decision rule. Therefore, a set of rules is obtained by a trained Decision Tree that is used for classifying unseen samples. An example of the Decision Tree approach to detect a BGP anomaly is shown in Fig. 5.3. In the example, the data set consists of regular and anomalous data with three features: number of announcement, number

of EGP packets, and number of AS path lengths. The input data points are categorized by the number of announcement and are divided into three groups: (1) Data points with the number of announcements smaller than 100 are classified as regular data points; (2) Data points with the number of announcements between 100 and 1000 are sorted by the number of EGP packets. Samples with the number of EGP packets < 4 are classified as regular; (3) Data points with the number of announcements larger than 1000 are sorted by the AS path lengths. Samples with the AS path lengths < 17 are classified as regular. The procedure repeats until the bottom of the tree is reached. The ellipses with classification results are called leaf nodes. The straight lines between layers of the tree represent branches. The same classification result applies to each new instance that follows the exact same branch. In practice, proper pruning of the tree is the key to achieving the desirable result. Otherwise, the tree may cause overfitting and lead to a poor result.

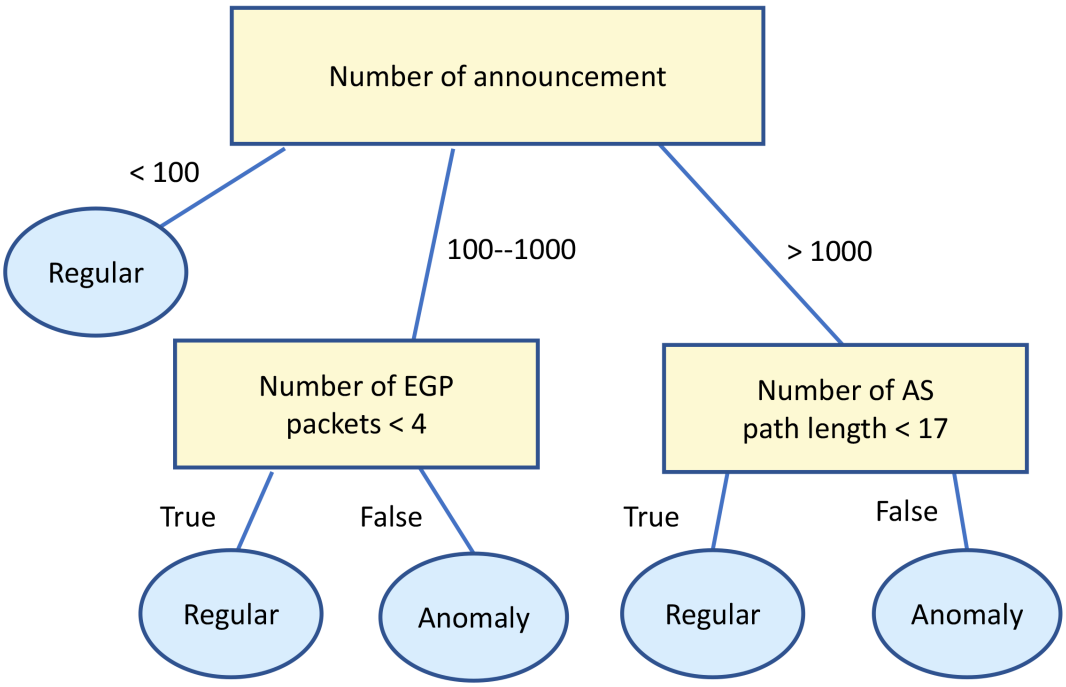

Figure 5.3: An example of the Decision Tree used to detect a BGP anomaly. The input data points are categorized based on the features shown in rectangles. The classification results are shown by ellipses that represent leaf nodes.

Entropy and information gain are key criteria for feature selection during the tree splitting. Entropy measures the level of impurity in a dataset. If we are given a training dataset X, entropy is calculated as [103], [104]:

$$E(X) = \sum_{c \in C} -p(c) log_2 p(c), \tag{5.13}$$

where c is a class label that belongs to a set of classes C. The proportion of a sample from data set X that belongs to class c is denoted as $p(c)$. The higher the entropy, the better dataset it is for training. For example, $E(X) = 0$ implies that all samples in X belong to the same class. For binary classification, $E(X) = 1$ implies that training samples with 50% in each class, which is desirable for classification.

The information gain measures the importance of features and determines how to split the Decision Tree. It is calculated based on the entropy:

$$InformationGain = entropy(parent) - average[entropy(children)] \tag{5.14}$$

or:

$$IG(a, X) = E(X) - \sum_{s \in S} p(s)E(s), \tag{5.15}$$

where IG represents the information gain. We denote the original dataset as X while a feature of the data is a. $IG(a, X)$ implies that the dataset is split based on the feature a. $E(X)$ is the entropy of the original dataset. s is a subset that belongs to a set of subsets S. We denote the proportion of the subset s to the number of total samples in X as $p(s)$. The entropy of the subset is denoted as $E(s)$.

C4.5 [101] software is a well known algorithm and software tool to generate a Decision Tree. As an extension from C4.5, C5.0 software, which was introduced in 1997, handles big datasets faster and more efficiently. C5.0 was used to generate Decision Tree [102] due to various advantages [105]:

- Boosting technique generates multiple classifiers that are combined together to vote for a final classification result. It supports Boosting to improve the prediction accuracy.
- It applies variable misclassification costs to minimize the costs for misclassification.
- Several new data types are permitted in C5.0 such as date, time, and case labels. Furthermore, it enables missing values to be noted as "not applicable".
- It contains additional functions such as sampling and cross-validation.
- Scalability is enhanced by multi-threading and may be used for computers with multiple CPUs and cores.

5.4 Extreme Learning Machine (ELM) Algorithm

The Extreme Learning Machine (ELM) [32], [106] is an efficient learning algorithm implemented with a single hidden layer feed-forward neural network. It randomly initializes the weights of the hidden layer and analytically determines the output weights. It is capable of universal approximation of any non-constant piecewise continuous function. ELM avoids

the iterative tuning of the weights used in traditional neural networks and, hence, it is fast and may be used as an online algorithm.

ELM employs weights connecting the input and hidden layers with the bias terms randomly initialized while the weights connecting the hidden and output layers are analytically determined. Its learning speed is higher than the traditional gradient descent-based method. Reported research results indicate that ELM may learn much faster than SVMs. Hence, the ELM algorithm is suitable for applications that require fast response and for real-time predictions. Incremental and weighted extreme learning machines are variants of the typical ELM.

A neural network architecture of the ELM algorithm is shown in Fig. 5.4, where $[x_1, x_2, \ldots, x_d]$ is the input vector; d is the feature dimension; $f(.)$ is the activation function; W is the vector of weights connecting the inputs to hidden units; $[y_1, y_2, \ldots, y_m]$ is the output vector; and β is the weight vector connecting the hidden and output units.

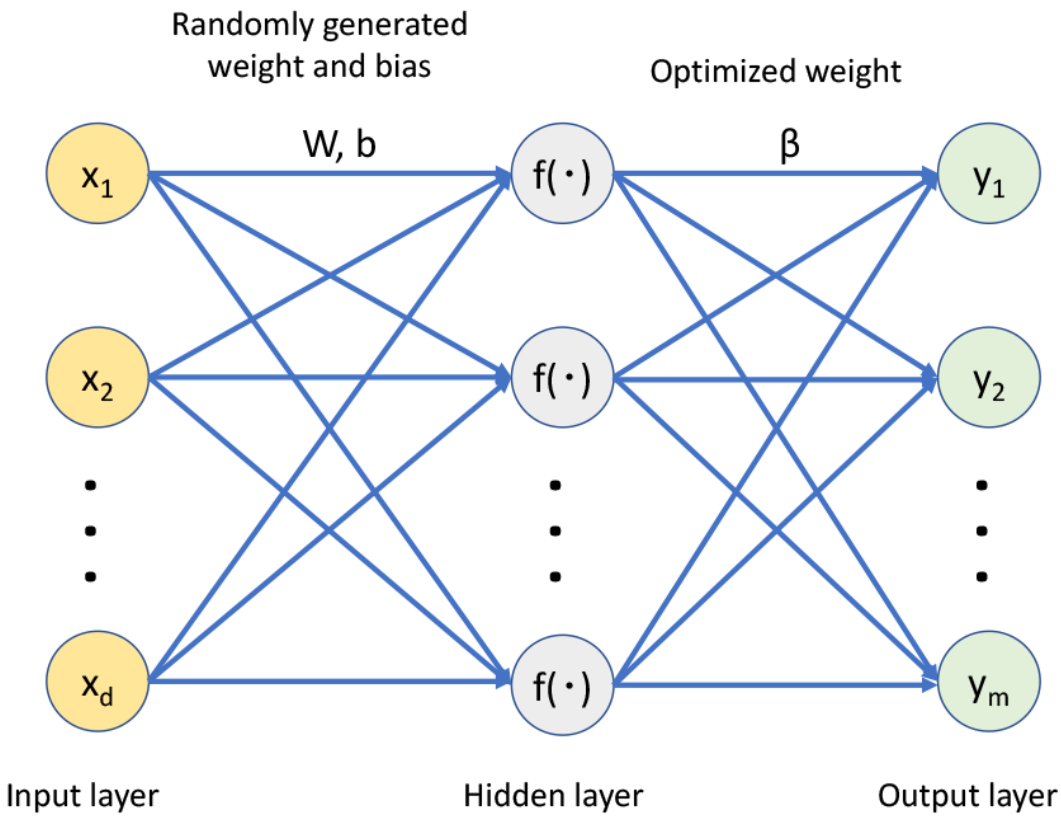

Figure 5.4: Neural network architecture of the ELM algorithm. Shown structure consists of an input layer with d dimensions, a hidden layer with k hidden neurons, and an output layer with m output samples.

In the case that the single-layered neural network performs a perfect approximation (the prediction error is zero), the output of the neural network with k hidden neurons is

calculated as:

$$\sum_{i=1}^{k} \beta_i f(w_i x_d + b_i) = y_m \tag{5.16}$$

or:

$$\mathbf{H}\beta = \mathbf{Y}, \tag{5.17}$$

where $\mathbf{H}$ denotes the hidden layer matrix that is defined as:

$$\mathbf{H} = \begin{bmatrix} f(w_1 x_1 + b_1) & \dots & f(w_k x_1 + b_k) \\ \vdots & \ddots & \vdots \\ f(w_1 x_d + b_1) & \dots & f(w_k x_d + b_k) \end{bmatrix}, \tag{5.18}$$

where the bias is $\beta = (\beta_1 \cdots \beta_k)^\top$ and the output is $\mathbf{Y} = (y_1 \cdots y_m)^\top$.

The three training and test datasets used in our previous study [75] to verify ELM's performance are shown in Table 2.3. The number of hidden units was selected by a 5-fold cross validation for each training dataset [75]. The best testing accuracy was achieved by using 315 hidden units for each dataset. Radial basis transfer function was chosen for the activation function. The input vectors of the training datasets were mapped onto $[-1, 1]$ as:

$$x_i^{(p)} = 2\frac{x_i^{(p)} - x_{i_{min}}}{x_{i_{max}} - x_{i_{min}}} - 1, \tag{5.19}$$

where $x_i^{(p)}$ is the i^{th} feature of the p^{th} sample while $x_{i_{min}}$ and $x_{i_{max}}$ are the minimum and maximum values of the i^{th} feature of the training sample, respectively.

5.5 Performance Comparison of Classification Algorithms

In a two-way classification, all anomalies are treated as one class. Validity of the proposed models is tested by applying two-way LSTM classification algorithm on BGP traffic traces collected from RIPE and BCNET on December 20, 2011. We then compare LSTM classification performance with SVM [12], Naïve Bayes, Decision Tree, and ELM models [60] reported in the previous study. The regular RIPE and BCNET datasets contain no anomalies and, hence, data points are labeled as regular traffic. Datasets listed in Table 2.3 are used to train the two-way classifiers. The test datasets are Code Red I, Nimda, Slammer, regular RIPE, and regular BCNET. The predicted models are estimated and validated by a 10-fold cross-validation. The exemption is the ELM models, which use 5-fold cross-validation.

Keras [87], a modular neural network library for the Python language, is designed for deep learning and used as a framework for implementing the LSTM classifier. It uses either TensorFlow or Theano library as the back-end. In this Thesis, we use Keras 2.0.2 with Python 2.7.13 and TensorFlow 1.0.1 [107]. We use all 37 features [12] because LSTM cells select the useful features during the learning process. LSTM models developed in the pre-

vious study [12] are improved by adjusting the length of time sequence and the choice of the optimizer.

5.5.1 Unbalanced Datasets

The training models are labeled based on the training datasets listed in Table 2.3. For example, the dataset used for the first set of unbalanced training models is a concatenation of Slammer and Nimda anomalies while the corresponding test dataset contains Code Red I anomaly. LSTM classifier was implemented using the unbalanced datasets with 37 features. We compare LSTM classification results to SVM, Naïve Bayes, Decision Tree, and ELM results reported in previous studies [12], [59], [60]. Comparison of classification results are shown in Table 5.1.

Table 5.1: Accuracy and F-Score using various classification models for unbalanced datasets.

Training model	**Test datasets**			
	Accuracy (%)			F-Score (%)
	Code Red I	RIPE regular	BCNET	Code Red I
$LSTM_u1$	95.22	65.49	57.30	83.17
SVM_u1	78.65	69.17	57.22	39.51
Naïve $Bayes_u1$	82.03	82.99	79.03	29.52
Decision $Tree_u1$	85.36	89.00	77.22	47.82
ELM_u1	80.92	75.81	69.03	36.27
	Nimda	RIPE regular	BCNET	Nimda
$LSTM_u2$	53.94	51.53	50.80	11.81
SVM_u2	55.50	89.89	82.08	24.29
Naïve $Bayes_u2$	62.56	82.85	86.25	48.78
Decision $Tree_u2$	58.13	94.19	81.18	26.16
ELM_u2	54.42	96.15	91.88	13.72
	Slammer	RIPE regular	BCNET	Slammer
$LSTM_u3$	95.87	56.74	58.55	84.62
SVM_u3	93.04	73.92	59.24	75.93
Naïve $Bayes_u3$	83.58	84.79	81.18	51.12
Decision $Tree_u3$	95.89	89.42	77.78	84.34
ELM_u3	86.96	78.57	73.47	55.31

In case of unbalanced datasets, the number of anomaly and regular data points for the training and test datasets are shown in Table 5.2. We use all 37 features for creating the models. Among the three models, $LSTM_3$ with Slammer test dataset exhibits the best F-Score for binary classification.

$LSTM_u1$ outperforms other classification algorithms with the best accuracy (95.22 %) and F-Score (83.17 %). Certain feature selection algorithms may slightly improve the performance of both SVM and Naïve Bayes, as reported in our previous studies [59], [60]. However, none of the algorithms performs better than LSTM for Training $Dataset_u1$.

Table 5.2: Number of anomalies in each unbalanced dataset.

Training dataset	**Anomaly data**	**Regular data**	**Total**
1	4,390	10,010	
2	1,469	12,931	14,400
3	4,121	10,279	
Test dataset	**Anomaly data**	**Regular data**	**Total**
1	869	6,331	
2	3,521	3,679	7,200
3	600	6,600	

Naïve Bayes$_u$2 has the highest accuracy (62.56 %) and F-Score (48.78 %). The overall performance of these algorithms is not desirable, which implies that none of the models is suitable for the analyzed dataset. We suspect the poor performance of the models using this training dataset is due to the distributions of anomaly and regular data points. Hence, the models may not be able to capture the relationships and the most relevant features among data points during the training phase.

LSTM$_u$3 achieves the best F-Score (84.62 %). Although Decision Tree$_u$3 achieves the best accuracy (95.89 %), LSTM$_u$3 shows competitive performance (95.87 %). The performance of SVM and Naïve Bayes was slightly better when using Fisher and Mutual Information Base (MIBASE) feature selection algorithms, as shown in our previous study [60].

We have also implemented a deeper LSTM model with two LSTM layers to train unbalanced datasets. However, the model was overfitting because of insufficient number of training data samples. Thus, these results have not been reported.

5.5.2 Balanced Datasets

We created three LSTM models using balanced datasets in order to compare their performance to SVM, Naïve Bayes, Decision Tree, and ELM results reported in previous studies [12], [60]. In case of balanced datasets, the number of anomaly and regular data points for the training and test datasets are shown in Table 5.3.

Table 5.3: Number of anomalies in each balanced dataset.

Training dataset	**Anomaly data**	**Regular data**	**Total**
1	4,390	4,390	8780
2	1,469	1,469	2,938
3	2,060	2,060	4,120
Test dataset	**Anomaly data**	**Regular data**	**Total**
1	869	6,331	
2	3,521	3,679	7,200
3	600	6,600	

Comparison of classification results are shown in Table 5.3. Among all models, the Decision Tree$_b$3 model achieves the best F-Score (77.60 %) and accuracy (93.38 %).

Table 5.4: Accuracy and F-Score using LSTM and SVM models for balanced datasets.

Training model	**Test datasets**			
	Accuracy (%)			F-Score (%)
	Code Red I	RIPE regular	BCNET	Code Red I
LSTM$_b$1	56.43	60.48	62.78	26.59
SVM$_b$1	64.03	76.57	88.61	24.36
Naïve Bayes$_b$1	78.68	63.04	78.19	30.95
Decision Tree$_b$1	75.78	82.31	76.60	32.27
ELM$_b$1	55.28	52.49	47.99	23.55
	Nimda	RIPE regular	BCNET	Nimda
LSTM$_b$2	56.32	44.27	53.58	65.96
SVM$_b$2	69.26	51.81	44.86	72.32
Naïve Bayes$_b$2	67.89	52.13	58.61	64.68
Decision Tree$_b$2	48.92	11.82	59.24	65.70
ELM$_b$2	63.74	34.90	57.85	65.92
	Slammer	RIPE regular	BCNET	Code Red I
LSTM$_b$3	82.98	55.00	48.20	58.54
SVM$_b$3	87.19	63.31	51.11	64.76
Naïve Bayes$_b$3	80.57	74.07	80.49	50.55
Decision Tree$_b$3	93.38	82.81	77.29	77.60
ELM$_b$3	72.78	51.32	40.90	42.25

We compare here performance of LSTM models for unbalanced and balanced training datasets. When using unbalanced datasets, LSTM$_u$1 and LSTM$_u$3 models achieve higher accuracy and F-Score than LSTM$_b$1 and LSTM$_b$3 models, respectively. Unbalanced datasets contain more data samples than balanced datasets and, hence, data points are better trained using the LSTM model. Furthermore, the number of anomalous data points is large enough to train the LSTM model. The model contains various gates, blocks, and layers, which requires a rather large number of training samples in order to avoid overfitting and misclassification.

Chapter 6

Discussion

The Border Gateway Protocol (BGP) is the most widely used inter-domain routing protocol. Over the past two decades, the Internet has been subjected to various types of malicious attacks such as the routing table leak and domain hijack. Anomaly detection is a challenging task because it is difficult to obtain labeled anomaly datasets. A dataset typically contains only a small portion of labeled anomalous data points, which may affect the classification result. One approach is to take regular data samples and randomly add artificial anomalies to create a new dataset. However, this may negatively affect the effectiveness of algorithms because the artificial datasets may not contain properties of the genuine data. Therefore, we used genuine data collected from RIPE and BCNET to avoid producing unrealistic results. We did not use Route Views datasets because they do not archive BGP data earlier than 2003 while Nimda and Code Red I occurred in 2001.

The selection of appropriate performance metrics is another challenge. Accuracy measures the ratio of true predictions to the entire dataset. Therefore, if all regular points are correctly classified while anomalies are all misclassified, the accuracy would be high because regular points are majority in unbalanced datasets. Therefore, we used the F-Score as the most relevant performance metric. It is based on precision and sensitivity and is more suitable for anomaly detection tasks because it emphasizes importance of the anomaly class.

We have introduced and compared the LSTM classifier to SVM, Naïve Bayes, Decision Tree, and ELM classifiers reported in our previous studies [12], [60] in order to examine the effectiveness of each approach for detecting network anomalies. Note that literature survey did not reveal similar studies using RIPE datasets. Hence, no comparison to other reported results was feasible.

This research project applies machine learning techniques to detect BGP anomalies and to illustrate that LSTM is a feasible approach. Each machine learning model evaluated in this project has its unique advantages and limitations. LSTM models evaluated in this Thesis achieved high accuracy and F-Score for time-sequential input data. However, they may not be suitable for random data samples. Soft-margin SVMs perform well in classification tasks. However, they require relatively long computational time for training models when

dealing with large datasets. Naïve Bayes algorithm computes probabilities of the occurrence of events and are suitable for detecting multiple classes of anomalies. Decision tree is commonly used in data mining due to its explicit and efficient decision making. ELM is an efficient classifier while its performance is limited due to its simple structure. Furthermore, in practice, selection of anomaly detection approaches is application dependent.

Chapter 7

Conclusion and Future Work

In this Thesis, we have classified anomalies in BGP traffic traces using a number of classification models. We conducted experiments using a number of datasets and various features extracted from these datasets. We compared performance of BGP anomaly detection models based on the LSTM, SVM, Naïve Bayes, Decision Tree, and ELM classifiers. The performance of classifiers is greatly influenced by the employed datasets. While no single classifier performs the best across all used datasets, machine learning has been shown to be a feasible approach to successfully classify BGP anomalies using various classification models.

Datasets used in this project were collected by RIPE and BCNET. In further studies, additional data sources such as Center for Applied Internet Data Analysis (Caida), KDD Cup 1999 data, Routing Assets Database (RADb) as well as additional types of anomalies such as prefix hijack attacks may be used to evaluate the performance of classifiers. In this Thesis, we concatenated two anomalies to generate training datasets and used the third anomaly as the test dataset. The models may have performed better if one anomaly was appropriately partitioned and used for both training and testing. A different approach may be to concatenate three anomalies to form a larger dataset and then partition it into the training and test datasets in order to detect one anomaly. This method may help machine learning algorithms to identify additional characteristics of the datasets and relationships among data points.

LSTM is a relatively new technique and it requires further studies to examine its advantage and limitations for detecting BGP anomalies. Additional techniques may be used to optimize the LSTM performance. For example, dropout technique may reduce noisy data points by randomly discarding a certain number of data samples and, thus, increase the computational efficiency of LSTM models. By using dropout technique in the input layer, the network may learn additional internal relationships among data samples. Because the dropout layer eliminates some input samples, it results in other samples replacing missing ones in order to make predictions. This implies that the prediction model has to learn new patterns from the input dataset. The convergence efficiency may be improved by tuning

hyperparameters such as the number of LSTM cells in each LSTM layer and the number of epochs during the training phase.

Bibliography

[1] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, *IETF*, Mar. 1995. [Online]. Available: `http://tools.ietf.org/rfc/rfc1771.txt` [June 2018].

[2] D. P. Watson and D. H. Scheidt, "Autonomous systems," *Johns Hopkins APL Technical Digest*, vol. 26, no. 4, pp. 368–376, Oct.–Dec. 2005.

[3] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," RFC 4271, *IETF*, Jan. 2016. [Online]. Available: `http://tools.ietf.org/rfc/rfc4271.txt` [June 2018].

[4] T. G. Griffin and B. J. Premore, "An experimental analysis of BGP convergence time," in *Proc. ICNP, Riverside*, CA, USA, Nov. 2001, pp. 53–61.

[5] J. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th edition)*. J. F. Kurose, K. W. Ross, Eds, Addison-Wesley, 2012, pp. 305–431.

[6] YouTube Hijacking: A RIPE NCC RIS case study [Online]. Available: `http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study` [June 2018].

[7] K. Fukushima, "Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol Cybern*, vol. 36, pp. 193–202, 1980.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag, 2006, pp. 325–358.

[9] T. M. Mitchell, "Decision tree learning," in *Machine Learning*, Eric Munson Ed. WCB/McGraw-Hill, 1997, pp. 52–78.

[10] A. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950.

[11] G. E. Hinton, S. Osindero, and Y-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Compt.*, vol. 18, no. 7, pp. 1527–1554, July. 2006.

[12] Q. Ding, Z. Li, P. Batta, and Lj. Trajković, "Detecting BGP anomalies using machine learning techniques," in *Proc. IEEE Int. Conf. Syst., Man, and Cybern.*, Budapest, Hungary, Oct. 2016, pp. 3352–3355.

[13] R. Picard and D. Cook, "Cross-validation of regression models," *J. Amer. Statist. Assoc.*, vol. 79, no. 387, pp. 575–583, 1984.

[14] K. P. Burnham and D. R. Anderson, *Model Selection and Multimodel Inference (2nd ed.)*, New York, NY, USA: Springer-Verlag, 2002, pp. 32–35.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," [Online]. Available: `https://arxiv.org/pdf/1207.0580.pdf` [June 2018].

[16] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, Feb. 2012.

[17] A. Munoz and J. Moguerza, "Estimation of high-density regions using one-class neighbor machines," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 3, pp. 476–480, Mar. 2006.

[18] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *Proc. 26th IEEE Int. Conf. Comput. Commun.*, Anchorage, AK, USA, May 2007, pp. 625–633.

[19] T. Ahmed, B. Oreshkin, and M. Coates, "Machine learning approaches to network anomaly detection," in *Proc. USENIX Workshop on Tackling Comp. Syst. Problems with Mach. Learn. Techn.*, Cambridge, MA, Apr. 2007, pp. 1–6.

[20] J. Mao and A. K. Jain, "A self-organizing network for hyperellipsoidal clustering (HEC)," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp, 16–29, Jan. 1996.

[21] M. W. Richardson, "Multidimensional psychophysics," *Psychological Bulletin*, vol. 35, pp. 659–660, 1957.

[22] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.

[23] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[24] A. Raftery, "A note on Bayes factors for log-linear contingency table models with vague prior information," *Journal of the Royal Statistical Society*, vol. 48, no. 2, pp. 249–250, 1986.

[25] M. J. A. Berry and G. S. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Support.* Indianapolis, Indiana, USA: Wiley Publishing, Inc., 2004.

[26] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," *Expert Syst. Appl.*, vol. 36, no. 7, pp. 10206–10222, Sept. 2009.

[27] D. N. T. How, K. S. M. Sahari, Y. Hu, and C. K. Loo, "Multiple sequence behavior recognition on humanoid robot using long short-term memory (LSTM)," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, Hong Kong, China, Dec. 2014, pp. 109–114.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Oct. 1997.

[29] X. Yang, Q. Song, and A. Cao, "Weighted support vector machine for data classification," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Montreal, QC, Canada, Aug. 2005, vol. 2, pp. 859–864.

[30] D. Mladenic and M. Grobelnik, "Feature selection for unbalanced class distribution and naive Bayes," in *Proc. Int. Conf. Machine Learning*, Bled, Slovenia, June 1999, pp. 258–267.

[31] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[32] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, Dec. 2006.

[33] G. B. Huang, X. J. Ding, and H. M. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. 74, no. 1–3, pp. 155–163, Dec. 2010.

[34] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Comput.*, vol. 8, no. 7, pp. 1341–1390, Oct. 1996.

[35] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Rev.*, vol. 7, pp. 1–30, Jan. 2006.

[36] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Comput.*, vol. 10, no. 7, pp. 1895–1924, Oct. 1998.

[37] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, Oct. 2015.

[38] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning." *MIT Press*, 2016. [Online]. Available: `http://www.deeplearningbook.org` [June 2018].

[39] K. Cho, B. V. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," [Online]. Available: `https://arxiv.org/abs/1406.1078` [June 2018].

[40] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *Trans. Sig. Proc.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[41] D. P. Watson and D. H. Scheidt, "Autonomous systems," *Johns Hopkins APL Technical Digest*, vol. 26, no. 4, pp. 368–376, Oct.–Dec. 2005.

[42] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Commun. Surveys Tut.*, vol. 16, no. 1, pp. 303–336, Mar. 2014.

[43] F. Lau, S. H. Rubin, M. H. Smith, and Lj. Trajković, "Distributed denial of service attacks," in *Proc. IEEE Int. Conf. Syst., Man, and Cybern., SMC 2000*, Nashville, TN, USA, Oct. 2000, pp. 2275–2280.

[44] C. Patrikakis, M. Masikos, and O. Zouraraki, "Distributed denial of service attacks," *The Internet Protocol*, vol. 7, no. 4, pp. 13–31, Dec. 2004.

[45] H. Hajji, "Statistical analysis of network traffic for adaptive faults detection," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1053–1063, Sept. 2005.

[46] M. Thottan and C. Ji, "Anomaly detection in IP networks," *IEEE Trans. Signal Process.*, vol. 51, no. 8, pp. 2191–2204, Aug. 2003.

[47] S. Deshpande, M. Thottan, T. K. Ho, and B. Sikdar, "An online mechanism for BGP instability detection and analysis," *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1470–1484, Nov. 2009.

[48] K. El-Arini and K. Killourhy, "Bayesian detection of router configuration anomalies," in *Proc. Workshop Mining Netw. Data*, Philadelphia, PA, USA, Aug. 2005, pp. 221–222.

[49] J. Zhang, J. Rexford, and J. Feigenbaum, "Learning-based anomaly detection in BGP updates," in *Proc. Workshop Mining Netw. Data*, Philadelphia, PA, USA, Aug. 2005, pp. 219–220.

[50] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal, "An Internet routing forensics framework for discovering rules of abnormal BGP events," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 55–66, Oct. 2005.

[51] C. Wagner, J. Francois, R. State, and T. Engel, "Machine learning approach for IP-flow record anomaly detection," in *Lecture Notes in Computer Science: Proc. 10th Int. IFIP TC 6 Netw. Conf.*, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, C. Scoglio, Eds. Springer 2011, vol. 6640, pp. 28–39.

[52] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. Eur. Symp. Artificial Neural Netw., Comput. Intell. Mach. Learn.*, Bruges, Belgium, Apr. 2015, pp. 89–94.

[53] M. Cheng, Q. Xu, J. Lv, W. Liu, Q. Li, and J. Wang, "MS-LSTM: a multi-scale LSTM model for BGP anomaly detection," in *Proc. 2016 IEEE 24th Int. Conf. Netw. Protocols, Workshop Mach. Learn. Comput. Netw.*, Singapore, Nov. 2016, pp. 1–6.

[54] D. Ariu, R. Tronci, and G. Giacinto, "HMMPayl: an intrusion detection system based on Hidden Markov Models," *Comput. Security*, vol. 30, no. 4, pp. 221–241, 2011.

[55] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. Int. Conf. Measurement and Modeling of Comput. Syst.*, Banff, AB, Canada, June 2005, pp. 50–60.

[56] W. Zong, G. B. Huang, and Y. Chen, "Weighted extreme learning machine for imbalance learning," *Neurocomputing*, vol. 101, pp. 229–242, Feb. 2013.

[57] K. Zhang, A. Yen, X. Zhao, D. Massey, S.F. Wu, and L. Zhang, "On detection of anomalous routing dynamics in BGP," in *Lecture Notes in Computer Science: Proc. Int. Conf. Research Netw.*, N. Mitrou, K. Kontovasilis, G. N. Rouskas, I. Iliadis, L. Merakos, Eds. Springer 2004, vol. 3042, pp. 259–270.

[58] PyBrain: The Python Machine Learning Library. [Online]. Available: http://pybrain.org/[June 2018]

[59] Q. Ding, Z. Li, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks: datasets and feature selection algorithms," inÂă*Cyber Threat Intelligence*, M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, pp. 47–70, 2018.

[60] Z. Li, Q. Ding, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks: classification algorithms," in *Cyber Threat Intelligence*, M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, pp. 71–92, 2018.

[61] Y. Li, H. J. Xing, Q. Hua, X. Z. Wang, P. Batta, S. Haeri, and Lj. Trajković, "Classification of BGP anomalies using decision trees and fuzzy rough sets," in *Proc. IEEE Trans. Syst., Man, Cybern.*, San Diego, CA, USA, Oct. 2014, pp. 1331–1336.

[62] Center for Applied Internet Data Analysis. The Spread of the Sapphire/Slammer Worm [Online]. Available: http://www.caida.org/publications/papers/2003/sapphire/ [June 2018].

[63] Sans Institute. Nimda Worm—Why Is It Different? [Online]. Available: http://www.sans.org/reading-room/whitepapers/malicious/nimda-worm-different-98 [Jan. 2017].

[64] Sans Institute. The mechanisms and effects of the Code Red worm. [Online]. Available: https://www.sans.org/reading-room/whitepapers/dlp/mechanisms-effects-code-red-worm-87 [June 2018].

[65] Sans Institute. Malware FAQ: MS-SQL Slammer. [Online]. Available: https://www.sans.org/security-resources/malwarefaq/ms-sql-exploit [June 2018].

[66] University of Oregon Route Views project [Online]. Available: http://www.routeviews.org/ [June 2018].

[67] RIPE NCC: RIPE Network Coordination Center. [Online]. Available: http://www.ripe.net/data-tools/stats/ris/ris-raw-data [June 2018].

[68] T. Farah, S. Lally, R. Gill, N. Al-Rousan, R. Paul, D. Xu, and Lj. Trajković, "Collection of BCNET BGP traffic," in *Proc. 23rd ITC*, San Francisco, CA, USA, Sept. 2011, pp. 322–323.

[69] S. Lally, T. Farah, R. Gill, R. Paul, N. Al-Rousan, and Lj. Trajković, "Collection and characterization of BCNET BGP traffic," in *Proc. 2011 IEEE Pacific Rim Conf. Commun. Comput. and Signal Process.*, Victoria, BC, Canada, Aug. 2011, pp. 830–835.

[70] MRT rooting information export format. [Online]. Available: http://tools.ietf.org/html/draft-ietf-grow-mrt-13 [June 2018].

[71] The Internet Engineering Task Force (IETF) [Online]. Available: https://www.ietf.org/ [June 2018].

[72] Bgpdump [Online]. Available: https://bitbucket.org/ripencc/bgpdump/wiki/Home [June 2018].

[73] BGPmon [Online]. Available: https://bgpmon.net/ [June 2018].

[74] BCNET. [Online]. Available: http://www.bc.net [June 2018].

[75] N. Al-Rousan and Lj. Trajković, "Machine learning models for classification of BGP anomalies," in *Proc. IEEE Conf. on High Performance Switching and Routing, HPSR 2012*, Belgrade, Serbia, June 2012, pp. 103–108.

[76] D. Meyer, "BGP communities for data collection," RFC 4384, *IETF*, Feb. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4384.txt [June 2018].

[77] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and analysis of BGP behavior under stress," in *Proc. 2nd ACM SIGCOMM Workshop on Internet Meas.*, New York, NY, USA, 2002, pp. 183–195.

[78] D. Blazakis and J. S. Baras, "Analyzing BGP ASPATH behavior in the Internet," in *Proc., 9th IEEE Glob. Int. Symp.*, 2006.

[79] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: special issue on learning from imbalanced data sets," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 1–6, June 2004.

[80] C. F. Lin and S. D. Wang, "Fuzzy support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 464–471, Feb. 2002.

[81] F. Provost, T. Fawcett, and R. Kohavi, "The case against accuracy estimation for comparing induction algorithms," in *Proc. 15th Int. Conf. Mach. Learn.*, Madison, WI, USA, July 1998, pp. 445–453.

[82] K. Morik, P. Brockhausen, and T. Joachims, "Combining statistical learning with a knowledge-based approach-a case study in intensive care monitoring," in *Proc. Int. Conf. Mach. Learn., ICML 1999*, Bled, Slovenia, June 1999, pp. 268–277.

[83] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, May 1992.

[84] "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," [Online]. Available: http://arxiv.org/pdf/1402.1128 [June 2018].

[85] Understanding LSTM Networks [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [June 2018].

[86] F. A. Gers, J. Schimidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol, 12, no. 10, pp. 2451–2471, Oct. 2000.

[87] Keras: Deep Learning library for Theano and TensorFlow. [Online]. Available: https://keras.io/ [June 2018].

[88] TensorFlow. [Online]. Available: https://www.tensorflow.org [June 2018].

[89] Deeplearning4j [Online]. Available: https://deeplearning4j.org/ [June 2018].

[90] Microsoft Cognitive Toolkit [Online]. Available: https://www.microsoft.com/en-us/cognitive-toolkit/ [June 2018].

[91] Theano [Online]. Available: http://deeplearning.net/software/theano/ [June 2018].

[92] A. Gulli and S. Pal, *Deep Learning with Keras*, Birmingham, UK: Packt Publishing Ltd. 2017, pp. 28–39.

[93] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: a simple way to prevent neural network from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[94] D. P. Kingma and J. Ba, "Adam: A method for sochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, USA, Dec. 2014.

[95] V. Vapnik, *The Nature of Statistical Learning Theory*, New York, NY: Springer-Verlag New York, Inc., 1995.

[96] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. 1997 Conf. Compt. Vision and Pattern Recognition*, San Juan, Puerto Rico, USA, June 1997, pp. 130–138.

[97] T. Joachims, "Text categorization with support vector machines: learning with many relevant features," in *Proc. 10th Eur. Conf. Mach. Learn.*, Chemnitz, Germany, Apr. 1998, pp. 137–142.

[98] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[99] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*, New York, NY, USA: Springer Publishing Company, Inc., 2014.

[100] X. Z. Wang, L. C. Dong, and J. H. Yan, "Maximum ambiguity based sample selection in fuzzy decision tree induction," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 8, pp. 1491–1505, Aug. 2012.

[101] C4.5 Tutorial. [Online]. Available: http://www2.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html [June 2018].

[102] Y. Li, H. J. Xing, Q. Hua, X.-Z. Wang, P. Batta, S. Haeri, and Lj. Trajković, "Classification of BGP anomalies using decision trees and fuzzy rough sets," in Proc. IEEE Int. Conf. Syst., Man, and Cyber. (SMC 2014), San Diego, CA, October 2014, pp. 1312-1317.

[103] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[104] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers–a survey," *IEEE Trans. Syst., Man, Cybern., Appl. and Rev.*, vol. 35, no. 4, pp. 476–487, Nov. 2005.

[105] C5.0: An Informal Tutorial [Online]. Available: `http://rulequest.com/see5-unix.html` [June 2018].

[106] Extreme Learning Machines. [Online]. Available: `http://www3.ntu.edu.sg/home/egbhuang/elm_codes.html` [June 2018].

[107] TensorFlow. [Online]. Available: `https://www.tensorflow.org/` [June 2018].

[108] Caida: Center for Applied Internet Data Analysis [Online]. Available: `http://www.caida.org/home/` [Aug. 2018].

[109] KDD Cup 1999 Data [Online]. Available: `hhttp://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` [Aug. 2018].

[110] RADb: The Internet Routing Registry [Online]. Available: `http://www.radb.net/` [June 2018].

[111] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," [Online]. Available: `https://arxiv.org/abs/1412.3555` [June 2018].

Appendix A

Script of the Long Short-Term Memory model used to classify BGP datasets

The implemented LSTM model contains an input layer with 37-dimensional input nodes, a hidden layer with 256 LSTM cells, a dropout layer with 50% dropout rate, and an output layer. The hidden layer uses the *ReLU* activation function while the output layer uses the *sigmoid* function. We use the "Adam" optimizer with the learning rate "lr = 0.001" to calculate the gradients and update the weight when compiling LSTM models. We set the value of the random seed to 77 in order to ensure the results are reproducible. We set the batch size to 32, which implies 32 instances are trained together and 32 predictions are made each time. We use 20% of the original training dataset to validate the model. The parameters that lead to a desirable classification result are saved and are then applied in the test phase. To avoid overfitting, 30 trials are made for all tested datasets.

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense, Activation
from sklearn.preprocessing import MinMaxScaler

# Fix the random number seed to ensure our results are reproducible.
num_feature = 37
look_back = 19
num_epoch = 30
# Load the dataset, X: data, Y: label
dataset = numpy.loadtxt("/Users/Desktop/data_1.csv", delimiter=",")
origin_X = dataset[:, 0:num_feature]
origin_Y = dataset[:, num_feature]
origin_X = origin_X.astype('float32')
```

```
origin_Y = origin_Y.astype('int16')

testDataset = numpy.loadtxt("/Users/Desktop/data_2.csv", delimiter=",")
test_X = testDataset[:, 0:num_feature]
test_Y = testDataset[:, num_feature]
test_X = test_X.astype('float32')
test_Y = test_Y.astype('int16')

# Normalize the dataset
#X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
#X_scaled = X_std * (max - min) + min
scaler = MinMaxScaler(feature_range=(0, 1))
train_X = scaler.fit_transform(origin_X)
train_Y = origin_Y
test_X = scaler.fit_transform(test_X)
test_Y = test_Y

#Change labels -1 to 0
inds = numpy.where(train_Y == -1)
train_Y[inds] = 0
inds = numpy.where(test_y == -1)
test_Y[inds] = 0

# Convert an array of values into a matrix
def create_dataset(X, Y, look_back=1):
dataX, dataY = [], []
for i in range(len(X) - look_back - 1):
a = X[i:(i + look_back), :]
dataX.append(a)
dataY.append(Y[i + look_back])
return numpy.array(dataX), numpy.array(dataY)

# Reshape the dataset into X=t and Y=t+1
trainX, trainY = create_dataset(train_X, train_Y, look_back)
testX, testY = create_dataset(test_x, test_y, look_back)
print trainX.shape
print trainY.shape

# Reshape the input to be [samples, time steps, features]
trainY = numpy.reshape(trainY, (trainY.shape[0], 1))
print trainY.shape

# Create and validate the LSTM network
model = Sequential()
model.add(LSTM(256, input_shape=(look_back, num_feature), \
activation='relu', return_sequences=True))
model.add(Dropout(0.5))
```

```
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', \
metrics=['accuracy'])

numpy.random.seed(77)
model.fit(x=trainX, y=trainY, epochs=num_epoch, batch_size=37, \
verbose=2, validation_split=0.2, shuffle = true)
model_score = model.evaluate(testX, testY)
print("validation accuracy: %.2f%%" % (model_score[1]*100.0))

model.save('model_lstm.h5')
model = load_model('model_lstm.h5')

#Apply the model to test the dataset
trainPredict = model.predict(trainX)
test_pred = model.predict(testX)

trainPredict_rounded = [round(x[0]) for x in trainPredict]
train_f1 = f1_score(trainY, trainPredict_rounded)
print("Train F-score: %.2f%%" % (train_f1*100.0))

test_score = model.evaluate(testX, testY)
print("Test accuracy: %.2f%%" % (test_score[1]*100.0))

test_pred_rounded = [round(x[0]) for x in test_pred]
f1 = f1_score(testY, test_pred_rounded)
print("Test F-score: %.2f%%" % (f1*100.0))
```