2022

A Predictive Model to Predict Cyberattack Using Self-Normalizing Neural Networks

Oluwapelumi Eniodunmo
eniodunmo@marshall.edu

A PREDICTIVE MODEL TO PREDICT CYBERATTACK USING SELF-NORMALIZING NEURAL NETWORKS

A thesis submitted to
the Graduate College of
Marshall University
In partial fulfillment of
the requirements for the degree of
Master of Arts
in
Mathematics
by
Oluwapelumi Eniodunmo
Approved by
Dr. Raid Al-Aqtash, Committee Chairperson
Dr. Anna Mummert
Dr. Laura Adkins

Marshall University
December 2022

APPROVAL OF THESIS/DISSERTATION

We, the faculty supervising the work of Oluwapelumi Eniodunmo, affirm that the thesis, *A predictive model to predict cyberattack using Self Normalizing Neural Networks*, meets the high academic standards for original scholarship and creative work established by the Department of Mathematics and the College of Science. This work also conforms to the editorial standards of our discipline and the Graduate College of Marshall University. With our signatures, we approve the manuscript for publication.

Dr. Raid Al-Aqtash, Department of Mathematics	Committee Chairperson	11/7/2022 Date
Dr. Anna Mummert, Department of Mathematics	Committee Member	11/7/2022 Date
Dr. Laura Adkins, Department of Mathematics	Committee Member	11-7-2022 Date

Marshall University
College of Science
Department of Mathematics

One John Marshall Drive
Huntington, WV 25755-2560
Tel: 304-696-6482
Fax: 304-696-4646
marshall.edu

ACKNOWLEDGEMENTS

I thank almighty Allah for allowing me to make this a reality, as I have always been interested in machine learning, and myself for putting in the work. I also give special thanks to my parents and siblings for their support right from my primary education, they never stopped giving me their support. A big thank you to my supervisor Dr. Raid Al-Aqtash for accepting to work with me on this, and I cannot thank him enough for helping me with this and for the impactful knowledge he has shared with me during the course of this project. My appreciation also goes to my thesis committee members, Dr. Anna Mummert and Dr. Laura Adkins, whom I admire for their vast knowledge of mathematics and statistics, and their proficient clarity in explaining concepts. My motivation also stems from having an outstanding mentorship from Dr. Carl Mummert, he has significantly influenced me during my graduate program as he advises, guides, and helps me with challenges during my graduate program. I also highly recognize an outstanding mentor, Memo Rodriguez, who was my manager during my internship at Apple. His versatility and team spirit has always motivated me to keep improving myself and learning new things.

I dedicate this work to my family, Mary Adeyemo, Muna Lentison, and Ashefon Ayodele. I also dedicate this project to the late rapper Kirsnick Khari, popularly known as TakeOff, whose 2022 summer hit song titled *"us vs. them"* was the title of my project folder for this project before his demise.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Cyberattack is a never-ending war that has greatly threatened secured information systems. The development of automated and intelligent systems provides more computing power to hackers to steal information, destroy data or system resources, and has raised global security issues. Statistical and Data mining tools have received continuous research and improvements. These tools have been adopted to create sophisticated intrusion detection systems that help information systems mitigate and defend against cyberattacks. However, the advancement in technology and accessibility of information makes more identifiable elements that can be used to gain unauthorized access to systems and resources. Data mining and classification tools such as K-Nearest Neighbors, Support vector machines, and Decision trees, among others, have been improved over time and used to build models for intrusion detection systems. This enables information systems, internet-connected devices, or devices running on a computer network to gain immunity against cyberattacks. However, these classification models hit some limitations as the sample size of data increases. Neural networks is an artificial intelligence tool that has been in active research over recent years. It has proven to handle big data and understand complex relationships better than the previous classification methods. Recent studies have demonstrated to build better models by showing better accuracy for intrusion detection systems using neural networks. In this thesis, we use a class of neural networks known as Self-Normalizing Neural Networks, which implements a scaled exponential linear unit activation function (SELU) developed by Klambauer et al. [12], to build a predictive model to detect cyberattacks against normal network traffic or connections using classification, in the KDD CUP 99 dataset from the Third International Knowledge Discovery and Data Mining Tools Competition, that was held in 1999. The accuracy and precision of the self-normalizing neural networks is compared with that of the k-nearest neighbors and support vector machines. The self-normalizing neural network appears to perform better. It is an excellent classifier for denial-of-service attacks, probe attacks, and user-to-root attacks while efficiently predicting normal connection. The result in this thesis is

also compared with existing literature which appears to perform better.

CHAPTER 1

INTRODUCTION

With the advancements of technology over the years and its improvements from generation to generation, the possibilities of technology and its equipment are endless. Computer systems and technology have evolved from huge devices sitting in one spot to portable devices or gadgets that individuals use in their everyday lives. This includes mobile phones, tablets, electric cars, internet-powered refrigerators, and so much more. As these devices have become a massive part of human lives, we entrust our information and our daily activities to them. These devices have been created with security features that help restrict unauthorized access to the information we store on them. We have trusted these devices so well that governments, large privately owned businesses and individuals keep the most sacred and secretive information on them, allowing everyone to securely access information from anywhere at any time. Devices can communicate with each other, and it is possible to access remote information with these devices via a network. A network enables connection and information exchange between devices. This network which powers communication consists of 3 significant parameters, namely connection(or basic) features, content features, and traffic features [3]. Networks transfer different information between devices. Hence, they must be implemented with information security in consideration. Security implementations are usually in the form of but not limited to, firewalls, encrypted data, password-enabled systems, and multi-factor authentication. These security barriers can still be breached if sensitive security information that guards these devices is exposed. For example, firewalls are hardware or software network security guards that monitor incoming and outgoing network traffic against some set of predefined security rules. Connections or packets violating such rules are dropped. However, attacks such as encrypted injection attacks, where cyber criminals use phishing emails as a decoy to enable client users to run some hidden special program, is possible. Structured Query Language (SQL) injection attacks are a common hacking technique to breach through firewalls and password-enabled systems. This is achieved by running SQL commands through packets sent through a firewall, or passing commands through

credentials, to retrieve sensitive information from databases such as credit cards, or destroy database and system resources. Encrypted data is a form of security on data that requires a key to be decrypted. This could be compromised if the key is not kept safe or is exposed during information exchange and is intercepted during network traffic communication. The use of technical shortcomings of a network security mechanism to either gain access to unauthorized information or disrupt the elements of a network is known as a cyberattack [4]. The purpose of cyberattacks varies, including gaining access to unauthorized information, destroying or consuming system resources, and preventing legitimate users from gaining access. The process of an attack begins with a malicious user trying to obtain some unauthorized information from a destination computer, usually known as a host, using a computer known as the source over a network connection. The attack can either happen during the transfer of information between the client (source) and the destination (host) or directly from the cybercriminal.

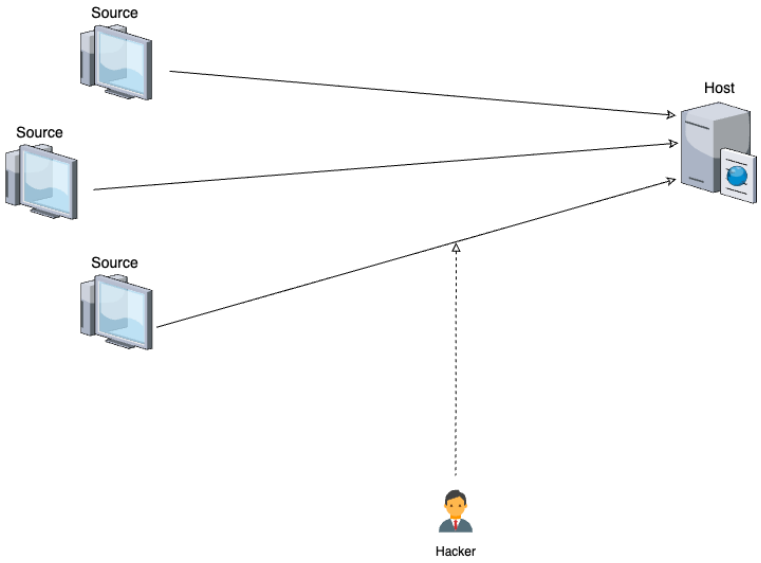

Figure 1.1: An attack on the host by a cybercriminal during information transfer

Figure 1.2: An attack directly to the host computer from a cybercriminal

There has been a trend of high-security breaches over the years. Well-known companies such as Facebook and Yahoo have been the target of cyberattacks. These companies have a vast amount of personal, geographical, and universal data that have been securely entrusted and stored. In October 2016, a distributed denial of service (DDOS) attack was executed by overwhelming different servers to disrupt network traffic, which caused a massive outage on the East Coast of the United States botnet controlled by a malware named Mirai. In May 2017, a ransomware software called WannaCry infected many computers by exploiting system vulnerabilities in the Windows operating system. It encrypted data and demanded a ransom in the form of bitcoin cryptocurrency to release user data [14]. Usually, some bits of information are sent over the network that can be used to get some unauthorized privilege to access or manipulate data at the destination or throw the destination computer into a state that makes it misbehave and not function as well as it is supposed to. While these bits of information are being transferred over the network connection, various parameters about the whole system can be studied or investigated to detect an attack, such as the connection parameters, information being transferred, operations being performed at the destination computer, and so many more. These security issues can be addressed by building an intrusion detection system (IDS), a well-known technology utilized to identify malicious connections or traffic to a target network system. IDS systems identify attacks without human intervention and have been developed over time machine learning algorithms have been a core part of analyzing and identifying various attack scenarios, however, with the emergence of new and more complex attack scenarios, machine learning based techniques have struggled to deal with such phenomenon.

The general idea of machine learning involves using a set of tools to understand data and

observe the relationship between a set of predictors $X = (x_1, x_2, ..., x_p)$ and a response Y, with the goal of estimating a function f which can be written in the general form

$$Y = f(X) + \epsilon,$$

where ϵ is an error term independent of X with a mean of zero. Machine learning can be categorized into supervised, unsupervised, and reinforcement learning. Regression and classification problems are supervised learning tools, and predictive models are assessed depending on the class of problems. Machine learning tools sometimes have a limitation in handling big datasets and studying complex relationships in data, but the area of deep learning, also known as neural networks, has been seen to handle such huge datasets and is capable of handling more complex relationships [9]. Neural networks began to gain popularity when convolution neural network (CNN) classified images correctly. Recurrent neural networks (RNN) also emerged, and is capable of analyzing and studying the relationship in speech patterns and conversations. Neural networks is a deep learning architecture that uses artificial neurons to process inputs in several hidden layers to build a predictive model from data and produce an optimal prediction.

Neural networks can be said to consist of 3 significant layers namely the input layer, the hidden layer, and the output layer.

The input layer typically consists of the features in the dataset. The hidden layer is responsible for the core operations of a neural network. It consists of 3 major parameters: a set of weights, bias and an activation function. The set of weights are used to emphasize or de-emphasize the importance of a predictor or neuron, in a neural network. The predictors and the weights are represented in matrix form, and then multiplied using matrix multiplication, as shown in equation 1.1,

$$N = \nu(X, W) = X \times W = x_1 \times w_1 + x_2 \times w_2 + \cdots + x_n \times w_n, \tag{1.1}$$

where X is a vector of the input data, W is a vector of the weights, which is used to emphasize and de-emphasize features of a dataset and N is the result of the matrix multiplication. Initially, the weights are generated randomly and then, the network learns the best weights for the data. A

bias can be added to a neural network after the weight is multiplied with the input. The role of a bias in a neural network is that it is the simple assumption that our model makes about our data. An activation function can then be applied to the result, which is then sent as a signal to the neuron in the next layer. Activation functions are usually non-linear functions, as they help to learn the non-linear relationship about our data. Some popular activation functions include the

1. Sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

2. Rectified Linear Unit function (ReLU),

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1.3}$$

3. Hyperbolic tangent function

$$H(x) = \tanh(x). \tag{1.4}$$

The output layer of a neural network typically consists of the output or predicted value. For a classification problem, the output of each observation will consist of a vector of probabilities for each target class of the problem, and any technique can be used, such as the highest probability, to choose the preferred class out of the vectors as the predicted value. Before the final expected value is given, neural networks perform the operations listed so far in two passes: a forward pass and a backward pass. The goal of neural networks is to fit a model in the form of a mathematical function to some data. The process maps observations from a dataset, known as inputs, to some desired predictions, known as the output, in the most optimal way. Thus, we want the predicted value to be as close as possible to the actual value. The forward pass involves several operations for calculating a predicted value, such as weight multiplication, the addition of a bias, and applying an activation function. At the end of a forward pass, the neural network calculates the

loss of the predicted value. For classification problems, the cross entropy loss function is used, and it is defined as

$$CE(p_i, y_i) = -y_i \times \log(p_i) - (1 - y_i) \times \log(1 - p_i),$$

where $i = 1, ..., n$, n is the number of classes in the classification problem, p_i and y_i are the predicted and actual probabilities of each class for an observation.

When the loss is computed, the forward pass is completed, but we are faced with the problem of minimizing the loss. The backward pass is responsible for minimizing the loss, and it is achieved by computing the partial derivative of the loss (L) with respect to the weights (W), given as $\frac{\partial L}{\partial W}$, in the neural network. This derivative is then used with a learning rate, which is a tuning parameter usually within 0.01 to 0.1, to update the weights which leads to a better prediction result. An optimizer determines how the weights are updated, by using a simple rule for updating the weights of a neural network based on the gradient of the loss and parameters in the network. There are several optimizer such as the stochastic gradient descent (SGD), adaptive momentum estimate (Adam) and mini batch gradient descent. For example, the stochastic gradient descent has its rule for updating the weights of a network defined as

$$W = W - \text{learning rate} \times \frac{\partial L}{\partial W}, \tag{1.5}$$

where W is the weight vector and L is the loss computed during the forward pass [22].

The process of supplying the whole dataset, calculating the loss and updating the weights, is called an epoch. Several epochs are required to obtain a better loss value, however, neural networks can learn with lesser epochs if we split our dataset into batches. A batch is a chunck of a training dataset that will be used for training a network, for an optimal weight value. Several chunks that contain the entire training dataset is used in each epoch training of a neural network. The size of a chunk is called a batch size.

Some common problems of neural networks are overfitting, vanishing gradient and exploding gradient. Overfitting occurs when a model is highly tailored to a training data such that, introducing a new training observation, makes a huge change to the model. There are various techniques for avoiding overfitting in neural networks, the very common ones are

1. Early stopping technique: it involves stopping the training process once the loss no longer improves.

2. Dropout technique: it involves randomly setting some neurons to zero based on a probability value.

The vanishing gradient problem is a case where the derivative of the loss with respect to the weights, $\frac{\partial L}{\partial W}$, is almost zero, due to the derivative of certain activation functions. For example, the derivative of the sigmoid function in equation 1.2, given as

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}, \tag{1.6}$$

tends to zero as the input gets large. Activation functions like the ReLU function in equation 1.3 does not face the vanishing gradient problem since its derivative

$$ReLU'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{1.7}$$

is either 0 or 1. However, the problem with ReLU is that when most of the inputs to the derivative are less than 0, the weights will not get updated. This is known as the dead ReLU. The exploding gradient problem is the other way around of the vanishing gradient problem, that is, the derivative of the loss with respect to the weights increases so fast, and the weights changes drastically during training. This makes it difficult for the model to obtain optimal weight values.

Deep learning models are increasingly being used because of their ability to handle big data and their better performance compared to traditional based machine learning solutions. We therefore take a look at how a class of neural networks can help us in detecting a cyberattack. The field of neural networks has been effective in studying the relationship in cyberattacks and it is a great tool for building intrusion detection systems. In 1998, Stoflo et al. of Lincoln labs, USA, ran a 7 week of network traffic to get a dataset consisting of compressed raw (binary) TCP-dump data of about 4GB. This data is then processed to the KDD Cup-99 data set which was used in the third international knowledge discovery and data mining tools contest. The KDD

Cup-99 dataset contains over 4 million records of connection vectors, each with 41 features, and are labeled as either a normal connection or a form of network attack with precisely one specific attack type. There are 22 attack types, and each falls into one of the four major categories [19]:

1. Denial of Service (DOS): The cybercriminal overloads the network with packets to the destination computer that consumes a huge amount of system resources or computing power, making the destination computer unavailable for legitimate users.

2. Probe: This involves observing the physical implementation of the chips of a computer system, then using the information, such as IP addresses, port numbers, internal wires and state of a network, to directly access a destination computer for information.

3. UserToRoot (U2R): The cybercriminal gains access to a computer as a normal user and exploits system vulnerability to upgrade privileges as a root user, enabling access to protected operations and information.

4. RemoteToLocal (R2L): This kind of attack involves a cybercriminal, who does not have any form of authorized access to the destination computer, sending packets containing information, such as a guessed account password or some form of authentication data that can grant access to the destination computer for information.

When a predictive model is formed, its purpose is to be used to forecast the result of a set of inputs or data supplied to it. The predictive model of a neural network is usually a function $f(X)$, where X can be a vector of features whose outcome is to be predicted. Several neural network models can be formed for a dataset depending on parameters combined such as the features, epoch, learning rate of the network and batch size. However, we need to know which of these models is the best for a dataset. We discuss some terms and tools that is being used to describe the performance of a predictive model [6].

1. Confusion matrix: The confusion matrix is a popularly used metric for observing the performance of classification problems. It is a cross tabulation of the actual values in the test data and the predicted values from a model. From this table we can obtain true

positives (TP), true negatives (TN), false positives (FN) and false negatives (FN), which enables the calculation of the accuracy, precision and recall of a model.

2. True Positives (TP): is the number of observations of a class that a model's prediction confirmed that it is that class when in reality, it is. This represents the number of correct predictions of a sample.

3. True Negatives (TN): is the number of observations of a class that a model's prediction confirmed that it is not that class when in reality, it is not.

4. False Positive (FP): is the number of observations of a class that a model's prediction confirmed that it is that class when in reality, it is not.

5. False Negatives (FN): is the number of observations of a class that a model's prediction confirmed that it is not that class when in reality, it is.

6. Accuracy: is the ratio of correctly predicted classes to the total number of records. It represents how well the model can predict correct values generally, and it is given as

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}.$$

7. Precision: Precision: A precision value is always with respect to a class. It gives us the correctly predicted value out of all the model's predictions about a class. This is the ratio of the correctly predicted class by the model to the total number of classes predicted to be positive by the model. It is given as

$$precision = \frac{TP}{TP + FP}.$$

8. Recall: A recall value is always with respect to a class. It gives us the correctly predicted value out of all the actual values of a class. It is given as

$$recall = \frac{TP}{TP + FN}.$$

In this thesis, we study the 10% KDD Cup 99 dataset using a class of neural networks known as self normalizing neural networks to build a predictive model. This is achieved by preprocessing the data, performing feature extraction and feeding a batch of the data as training data into the neural network, then assessing the model using a test batch with reports on the model performance.

CHAPTER 2

LITERATURE REVIEW

Intrusion detection in cyber-physical systems, which are systems that operate independently and make decisions without the need for human intervention, is an increasing area of research. There is less human intervention in identifying and preventing cyberattacks in such systems. Thus, these systems defend against attacks using defense mechanisms such as intrusion detection systems. There is a focus on malware classification of various types of intrusion as each system has attacks common to its unique baseline. These systems are large-scale, as they handle and process massive real-time information. They are geographically dispersed as they need to ensure services are provided to users near their location to avoid longer network communication time. They are also heterogeneous, combining various specialized computing systems to perform tasks [16]. Examples of such systems are first responder systems, such as implantable medical devices, self-driving cars, unmanned aircraft, and smart grids. Historically, the capacity to detect certain kinds of network intrusions relies primarily on the type and quantity of attacks carried out. The parameters involved are difficult to get due to being legally harmful materials such as packets with administrator commands, phishing links to execute destructive administrative tasks, and dangerous database operations harmful to the stored data. Thus, data is often generated using various simulations and reverse engineering. The KDD Cup 99 cyberattack dataset has been widely studied as a binary and multiclass target variable dataset. When the target variable is considered a binary class, the two connection classes are normal and attack. When viewed as a multiclass, a connection can be normal, or the kind of an attack that can be categorized as shown in Table 2.1

Category	Attack
DOS	back, land, neptune, pod, smurf, teardrop
U2R	buffer_overflow, loadmodule, perl, rootkit
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster
Probe	ipsweep, nmap, portsweep, satan

Table 2.1: Categories of the different attacks in the KDD Cup 99 dataset.

There is no best choice of machine learning models. Thus, researchers do not just rely on deep learning. It can be observed that certain attack types can be correctly predicted by some supervised and unsupervised learning when certain features are considered. Al-Mamory et al. [2] took an introspect into determining the best way to classify and analyze the KDD Cup 99 data set to get high accuracy in the classification of attacks. They concluded that specialized detectors were needed to classify the various attacks. They studied 20 different classifiers for the four attack types, and it was found that rare attacks, such as the U2R, can be efficiently detected with a classifier such as the Multivariate Adaptive Regression Splines (MARS), Fuzzy Logic and Random Forest classifiers, with accuracy ranging from 92% to 96%. Clustering techniques, such as k-means, probabilistic, and hierarchical clustering, were better classifiers for DOS and Probe attacks with accuracy ranging from 72% to 96%.

Ogundokun et al. [18] employed a particle swam optimization feature extraction technique to select variables, which was modeled using KNN and Decision Trees. The KNN had a better accuracy of 99.60% and a detection rate of 96.20%, which outperformed Hoque et al. [10] in terms of a detection rate of 95%. However, this experiment was conducted using the normal connection and Neptune attack label.

Fu et al. [7] addressed the problem of low detection rate in network attacks and the need for extensive feature engineering by proposing a deep learning model for network intrusion detection with imbalanced data for traffic anomaly detection. They addressed the imbalanced data issue by using an adaptive synthetic sampling method to expand minority class samples, then used a

modified stacked autoencoder for dimensionality reduction. They then combine an attention mechanism and bi-directional long short-term memory (LSTM) network to build a predictive model with an accuracy of 90.73% on the NSL-KDD dataset, which is also the KDD Cup 99 dataset but with redundant data removed.

The availability of big data and the breakthrough of deep learning in 2010 has led researchers to study the KDD Cup 99 dataset using neural networks. Moradi and Zulkernine [17] analyzed this dataset using a three-layer Multilayer Perceptron (two hidden layers with 35 neurons in each), which is a class of neural networks together with an early stopping validation technique. Without the technique, they encountered an overfitting problem that produced an undesirable model as the model produced an accuracy of less than 80% when test data was fed into the neural network. The early stopping validation technique solved the overfitting problem and improved the test data's accuracy to more than 90%.

Pooja and Purohit [21] modeled the KDD CUP 99 dataset using a Bi-directional LSTM, a neural network consisting of 5 dense layers responsible for weight and bias optimization. This network uses a dropout technique to avoid overfitting and enables faster training. The softmax activation is used, which returns probabilities of each class of connection type, and the highest probability is used as the target prediction. The Adam optimizer used in training the network with 100 epochs allows for finding the best minimum gradient loss in the network. Different activation functions were used, with softmax being the optimal activation function with a model accuracy of 99.73%. Other activation functions include ReLU with a model accuracy of 78.56%, Sigmoid with 99.65%, and Tanh with 81.66%. They also compared their result with traditional machine learning algorithms, including but not limited to the logistic regression method, classification decision trees, KNN classification method, naive bayes, and SVM. The Bi-directional LSTM appears to have better accuracy.

Podder et al. [19] discussed the prevailing cyberattacks comprehensively in the field of the internet of things and the effectiveness of deep learning to manage these attacks. They emphasized how deep learning models have shown significant improvement over traditional machine learning-based solutions, signature-based methods, and rule-based methods in order to address cybersecurity problems by comparing 85 research results where most researchers have

focused on malware classification and detection of various types of intrusion in the network. These results have shown that there is no perfect or best model for classifying cyberattacks, as different methods producing great results use different datasets. However, certain areas are remarkable. The Restricted Boltzmann Machine (RBM) deep learning technique is the most utilized deep learning technique in cybersecurity due to the unavailability of a dataset whose observations have a corresponding response, and it has an accuracy of 97.11% on the KDD Cup 99 data. The Recurrent Neural Network (RNN) is another popular solution for a large range of cybersecurity challenges due to the fact that cyberattack data are mostly treated as time series data. The RNN method has an accuracy of 77.55% on the KDD Cup 99 dataset. The deep belief network (DBN) and the LSTM methods are other great solutions for classifying cyberattack data. They appear to have high accuracies of 93.49% and 99.8%, respectively, on the KDD Cup 99 dataset. The research concluded by encouraging researchers to venture more into deep learning to cover more attack vectors, even though introducing new techniques is frowned upon. This is because deep learning carries out its training in hidden layers, which seem like black boxes, and it is difficult to identify errors in attack detection, which leads to costs and hazards to companies. A model can falsely detect an attack, and cyberattack experts might waste time investing in a false error.

Adams et al. [1] investigated the KDD Cup 99 dataset using an artificial neural network. They considered five classes of attack, which include the four classes, namely DOS, Probe, U2R, Probe, and the fifth class, the Buffer Overflow attack. They modeled the dataset using an artificial neural network (ANN) with one hidden layer consisting of 9 neurons and the hyperbolic tangent activation function, but the output layer used the softmax activation function with cross-entropy loss to calculate the loss during training. The model had an accuracy of 99.1%. A convolutional neural network was used by Liu and Zhang [14] on the KDD Cup 99 dataset to build a multiclass network intrusion detection model. They first transformed the dataset into a 2-dimensional form, then the network was built with two convolution layers, and two pooling layers ran on 30 iterations, with a dropout probability of 0.05. The softmax activation function and the cross entropy loss are used with an Adam optimizer to obtain an accuracy of 98.02% on the dataset. The accuracy appears to be better than the other CNN models compared with other literature. Thus, their model obtains better detection of unknown attacks, better generalization, and good

detection ability, compared to several works of literature using other neural networks such as the long short-term memory recurrent neural network (LSTM-RNN), gated recurrent unit recurrent neural network (GRU-RNN) and deep belief network (DBN), on cyberattack datasets.

CHAPTER 3

PROPOSED MODEL

A self-normalizing neural network is employed to understand better the relationship in the KDD Cup 99 data set to build an intrusion detection model. The 10% KDD Cup 99 dataset was used in this thesis, and it is available publicly on http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. This dataset contains 494,021 records and 41 features; out of the 9 categorical features, 6 of them are binary values, and there are 32 numeric features. The first step was to take a descriptive and graphical overview of the dataset. Each of the features was observed for missing values. No missing values were found, and there were no significant outliers from the dataset due to the domain of the data; the authors have pre-processed it from its raw form into a CSV file, and since an attack might involve manipulation of network parameters in a non-definite pattern, we expect that our data ranges through extreme values. We then categorize the specific attacks into their respective categories: DOS, U2R, R2L, and Probe.

3.1 Feature Selection

Feature selection is essential to modeling classification problems as it helps to learn meaningful features and removes the unimportant or noisy parts of a data set. Since the specific attacks in the dataset are being categorized, our responses are no longer the specific attacks and the normal connection but the categories of the attack and the normal connection. Each of the features is then observed for a relationship with the response. The responses are label encoded and represented as integers, as we can only work with numeric values during the model-building process. The categorical features in the dataset are scored with respect to the response using the mutual information method, which is a statistical measure of the dependence between two variables by evaluating the reduction in entropy in transforming a dataset. The information gain for each random variable X is calculated as

$$I(X;Y) = H(X) - H(X|Y),$$

where $I(X;Y) \geq 0$ is the mutual information for X and Y, $H(X)$ is the entropy for X and $H(X|Y)$ is the conditional entropy for X given Y. The value of $I(X;Y)$ represents the strength of the relationship between feature X and the response Y. A significant positive number represents a very strong relationship, and a small positive value represents a weak relationship. If $I(X;Y) = 0$, then the variables are independent. Thus, the features with higher information gain $I(X;Y)$ is selected. We use the python function *SelectKBest* from the *sklearn.feature_selection* module. This function takes as parameters: the mutual information statistic, the number of features to select as parameters, a set of categorical inputs, and the response, which is an encoded categorical label. The function produces the number of features that tends to be highly correlated with the response by scoring each feature according to their importance to the response.

The quantitative features are observed for a significant relationship with the categorized labels by plotting a boxplot between each numeric variable and the categorized labels. While we have the results showing which features have a relationship with the categorized labels, we use the analysis of variance (ANOVA) method to obtain the essential numeric features in our dataset by scoring each feature with respect to the response. The ANOVA determines the existence of statistically significant differences among several group means. Hence, we test for statistically significant differences among the response classes for a feature in our dataset. We use the python function *SelectKBest* from the *sklearn.feature_selection* module, more theoretical details for the ANOVA method can be found in [5]. This function takes the ANOVA statistic, the number of features to select as parameters, a set of numeric inputs, and the response variable, which is an encoded categorical label. It was also ensured that the inputs were normalized with a mean of 0 and a standard deviation of 1. Using the p-value from the F-statistic computed from the ANOVA, the function produces the number of features that tends to be highly correlated with the response by scoring each feature according to their importance to the response.

3.2 Pre-processing

Now that we have the features that matter to the response variable, we prepare the selected features for modeling. The categorical features are one-hot encoded, and the numeric features are standardized with a mean of 0 and a standard deviation of 1.

3.3 Self Normalizing Neural Networks (SNN)

The class of neural networks, known as self-normalizing neural networks, is a simple class of neural networks that model structured data by using a set of linear layers and non-linear activation function, known as a *Scaled Exponential Linear Unit* (SELU), which is given by

$$f(x) = \lambda \begin{cases} x, & x > 0 \\ \alpha e^x - \alpha, & x \leq 0 \end{cases}$$

where $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$ are predetermined. The assumptions of a self normalizing neural network are:

1. The inputs must have a mean of 0 and variance of 1.
2. The network weights should be lecun normalized, that is, they should have a mean of 0 and variance of $\frac{1}{n}$, where n is the number of neurons in the hidden layer.
3. The SELU activation function is used.
4. If dropout is used, then the alpha-dropout; which involves dropping neurons to $-\lambda\alpha = -1.7581$ because their point of low variance is in the negative region, should be used.

The SELU activation function proposed by Klambauer et al. [12], is self-normalizing the weight, biases and activations of the neural network. This activation function keeps a constant variance in the network, the self normalizing property is realized by the parameters λ and α as they keep inputs from previous layers that already have their mean and variance predefined on an interval, in that same interval.

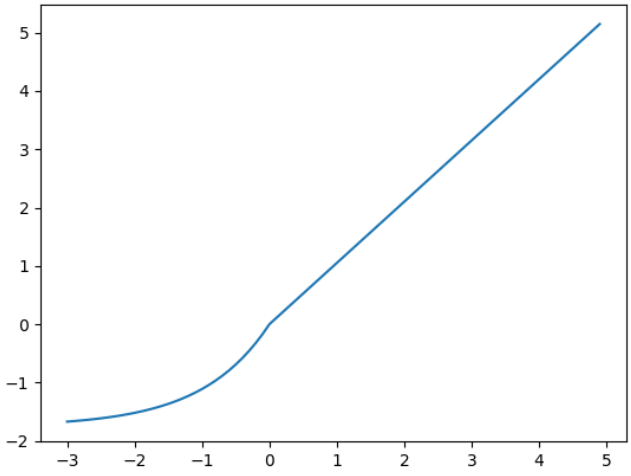

Figure 3.1: Graph of SELU activation function

The SELU function is special in that, it does not have the face the vanishing or exploding gradient problem, and it does not have the 'dead SELU' problem unlike the ReLU activation function.

3.4 K-Nearest Neighbors (KNN) for classification

The KNN for classification tool is a machine learning tool that is used to predict qualitative responses, given a positive integer K and a test observation x_0. To make a prediction, we first find the distance between existing observations and the new observation, any distance metric could be used such as the Minkowski distance. We then take the K training observations closest to x_0, where K is an odd positive integer. The conditional probabilities for each class j as the fraction of the set of K points, represented as N_0 is given as

$$\Pr[Y = j|X = x_0] = \frac{1}{k} \sum_{i \in N_0} I(y_i = j)$$

where $I(y_i = j)$ is an indicator variable that equals 1 if $y_i = j$, and 0 if $y_i \neq j$. Finally, KNN classifies the test observation x_0 to the class with the highest probability [11]. Although the KNN technique is simple and does not produce a model, but they simply classify observations without an existing response, by assigning them the class with the highest vote, amongst the observations

that have a response. However, the technique is very powerful as it is being used in computer vision applications and identifying patterns in genetic data for use in detecting specific proteins or diseases. The KNN method represents each observation on a scatter plot, then it locates the nearest neighbor using a distance metric. Any distance metric, such as the Minkowski and Chebyshev distance, can be used. The distance metric commonly used is the Euclidean distance given as

$$distance = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2},$$

where $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_n)$ represents the items to compare, each having n features. An odd value K is chosen and used to vote amongst the nearest neighbors of the new observation. The class with the highest vote is assigned to the observation. The choice of K plays a crucial role in KNN, as it balances overfitting and underfitting a model with its training data. A common approach to choosing K is to take the square root of the number of training sample sizes. If it is an even number, either one is added or subtracted to make it odd. Before applying the KNN to a dataset, it is important to scale the numeric features. One option of scaling is the min-max normalization given as

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)},$$

which transforms the data to have values between 0 and 1. Another option to scale the numeric features is to use the z-score standardization given as

$$X_{new} = \frac{X - \mu}{\sigma},$$

where μ is the mean and σ is the standard deviation of the feature X. It is important to scale the numeric predictors of a dataset, as the distance formula depends on how features are measured. Categorical data should be dummy encoded as integers. For example, a feature on two classes, male and female, can be represented with 0 and 1. A feature with n-nominal classes can be represented with $(n-1)$ binary indicator variables. For example, a feature on four classes, such as fruits, vegetables, proteins, and carbohydrates, can be represented, regardless of their order, with integers 1,2,3 and 4, respectively.

3.5 Support Vector Machines (SVM)

The support vector machines is a classifier that uses a hyperplane, an interpolating polynomial whose dimension is the number of predictors of a dataset, to classify observations. The hyperplane must be carefully chosen to avoid overfitting and the support vector machine solves this problem by using a maximal margin hyperplane. If X is a p-dimensional dataset, then support vector machines forms a hyperplane, represented as

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0.$$

If an observation satisfies the hyperplane, then $f(x) = 0$ otherwise, either $f(x) > 0$ or $f(x) < 0$. The maximal margin hyperplane defines the greatest minimum distance between observations and the separating hyperplane. The closest observations to the separating hyperplane are known as the support vectors. The width of the maximal margin hyperplane determines how much misclassification is allowed to be obtained by the model. We locate the positions of the misclassified observations represented as ϵ_i, and our goal is to find the best width for maximal margin hyperplane such that

$$\sum_i \epsilon_i \leq C,$$

where $C \geq 0$ is a tuning parameter for tuning the width of the hyperplane [13]. If $C = 0$, then no misclassification is accommodated, and this can lead to overfitting of the data. Formulating the hyperplane for the SVM classification requires an interpolating polynomial defined as the kernel of the SVM method, which includes linear, polynomial, radial basis function (RBF), and many more.

3.6 Evaluation Metrics

The confusion matrix is a popular metric for assessing a classification model's performance. It is a cross-tabulation of the actual labels in the test data and the predicted labels from a model. From this table, we can obtain true positives (TP), true negatives (TN), false positives (FN), and false negatives (FN), which enables the calculation of the accuracy, precision, and recall of our model.

CHAPTER 4

RESULTS

The KDD Cup 99 cyberattack dataset was imported into a python data frame and contained 494,021 records. The *labels* column contains 23 connection types, and each record is categorized into one of the four classes of attacks (DoS, U2R, R2L, Probe) or a normal category. We have imported the data to only include the best 24 features produced during our feature selection process, using the selectKBest function from the scikit-learn library as shown in table 4.1.

1. service	13. hot
2. loggedin	14. diffsrvrate
3. protocoltype	15. srvdiffhostrate
4. flag	16. samesrvrate
5. isguestlogin	17. dsthostsrvserrorrate
6. rootshell	18. serrorrate
7. count	19. srvserrorrate
8. dsthostcount	20. dsthostserrorrate
9. srvcount	21. rerrorrate
10. dsthostdiffsrvrate	22. dsthostsrvrerror_rate
11. dsthostsamesrcportrate	23. srvrerrorrate
12. dsthostsrvdiffhostrate	24. dsthostrerrorrate

Table 4.1: The 24 selected features

We selected 24 features to include features that fall into the three main categories listed in [15] while still maintaining the selections that best correlate with the target variable. The selectKBest function was used to select the top 6 categorical columns, using the mutual information method of selecting features, while the top 18 numeric columns were selected using the ANOVA F-value method. The categorical features were one-hot encoded. The quantitative variables were

standardized. The labels were also encoded using a label encoding scheme, as we need to represent our labels as integers. Then the data was split into train and test data so that we could train our model. The training and test data are 80% and 20% of the whole data, respectively.

A neural network consisting of an input dimension of 24, 5 layers of 30 neurons, and an output of 5 dimensions, since we have five classes of output, is constructed. We choose five layers as we want our neural network to learn the complex relationship in our dataset. We also want to choose the least layer efficient for a model, as the training time increases with the number of layers in a model. We also want to select at least 24 neurons since we have 24 features, and the number of layers and neurons are tuning parameters in a neural network. After several tunings for our model, a 5-layer and 30 neurons model performs efficiently for our dataset. Each layer performs the linear operations of weight multiplication and bias addition. It then applies the SELU activation function and uses the AlphaDropout technique with a dropout probability of 0.0002, which is the best value for our model after several tuning. The dropout probability is also a tuning parameter. Using a probability of 0.0002 means, there is a 0.0002 chance that an input neuron will be set to -1.7581, which is the dropout value for self-normalizing neural networks, instead of 0, like the traditional dropout technique. We use 500 epochs, a batch size of 10,000, and a learning rate of 0.05. Since this is a multiclass classification problem, we use cross-entropy loss to evaluate the loss in our model for each epoch and the Adam optimizer. The training data is then fed into the neural network model.

The performance of the model is evaluated on the test data. The SNN model can be seen in table 4.2 to have 99.97% accuracy in prediction with very high precision in detecting Normal connections, DoS attacks, and Probe attacks. Thus, our model is excellent for an intrusion detection system that intends to prevent such attacks. Our model is also great for R2L attacks with a high precision of 94%, but as for U2R attacks, we have a 70% precision. This is not bad for intrusion detection systems that do not primarily have U2R attacks.

Parameter tunings were made, such as selecting different combinations of 24 features that fall into the three main categories as listed in [15]. Batch sizes of 10,000, 20,000, and 60,000 were combined with the stochastic gradient descent and Adam optimizers. Table 4.3 shows some results of the different parameter tuning applied to the network.

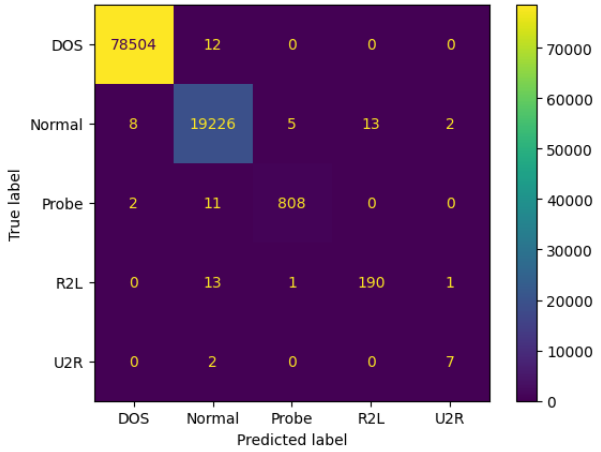

Figure 4.1: SNN Confusion Matrix for the 20% test data from the 10% KDD Cup 99 cyberattack dataset

	Class	Precision	Recall
SNN (99.97%)	Normal	100%	100%
	DoS	100%	100%
	Probe	99%	98%
	R2L	94%	93%
	U2R	70%	78%

Table 4.2: SNN report on the 20% test data from the 10% KDD Cup 99 cyberattack dataset

Batch Size	Learning Rate	Hidden Neurons	Dropout Probability	Optimizer	Accuracy
10,000	0.05	30	0.0002	Adam	99.97%
10,000	0.01	30	0	Adam	99.95%
20,000	0.01	32	0	SGD	99.48%
20,000	0.01	32	0.05	SGD	99.95%
60,000	0.01	30	0.001	SGD	99.94%

Table 4.3: Our neural network results with some of the different parameter tuning

The KNN algorithm is applied to the cyberattack dataset consisting of our selected features and also with 80% training data and 20% test data. Consequently, a 99.32% accuracy is achieved by choosing an odd k of 629, which is the square root of the number of training records in the KDD Cup 99 dataset. This does not perform as well as our SNN. The KNN can be seen in table 4.4 to have less precision for the attack classes than the SNN model. The SVM was also considered, and we obtained a 99.85% accuracy, which does not perform as well as the SNN. The SVM can be seen in table 4.5 to have less precision for the attack classes compared to the SNN model. It can be observed from table 4.4 that the KNN method has a precision of 0% for the U2R class of attack. This implies that both the KNN method fails to identify U2R attacks.

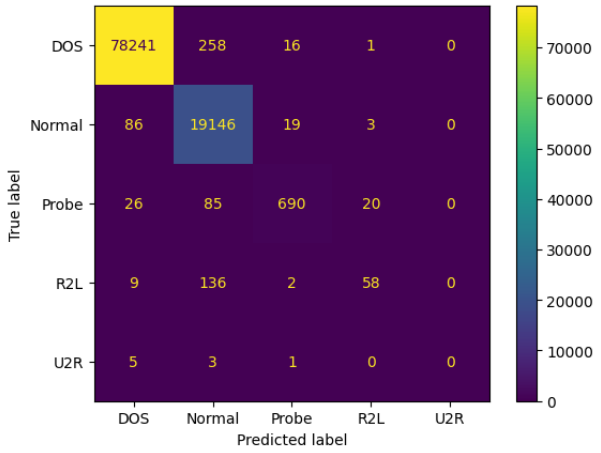

Figure 4.2: KNN Confusion Matrix for the 20% test data from the 10% KDD Cup 99 cyberattack dataset

	Class	Precision	Recall
KNN (99.32%)	Normal	98%	99%
	DoS	100%	100%
	Probe	95%	84%
	R2L	71%	28%
	U2R	0%	0%

Table 4.4: KNN report on the 20% test data from the 10% KDD Cup 99 cyberattack dataset

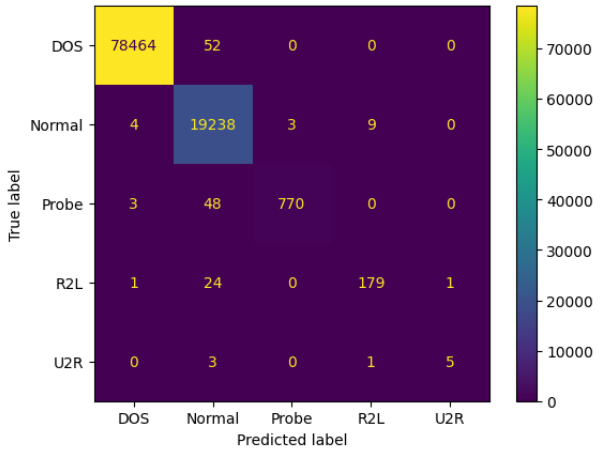

Figure 4.3: SVM Confusion Matrix for the 20% test data from the 10% KDD Cup 99 cyberattack dataset

	Class	Precision	Recall
SVM (99.85%)	Normal	99%	100%
	DoS	100%	100%
	Probe	100%	94%
	R2L	95%	87%
	U2R	83%	56%

Table 4.5: SVM report on the 20% test data from the 10% KDD Cup 99 cyberattack dataset

CHAPTER 5

CONCLUSION

In this thesis, we propose using a class of neural networks, known as self-normalizing neural networks (SNN), on the KDD Cup 99 cyberattack dataset to detect network intrusion. The response variable consists of 22 attack types categorized into four categories (DoS, U2R, R2L, and Probe) and one normal connection. The dataset consists of 41 predictors, and we chose 24 predictors for the modeling problem employing feature selection. The categorical predictors were scored according to their relationship with the response variable. The top 6 predictors with a strong relationship with the response variable were selected using the mutual information method. The quantitative predictors were also scored according to their relationship with the response variables. The top 18 predictors with a strong relationship with the response variable were selected using the ANOVA F-value method.

The response variable and the categorical predictors are integer encoded, and the numeric predictors are normalized before they are passed through the neural network. The self-normalized neural network is made up of 5 layers with 30 neurons. Each layer has its weights to be *lecun normalized*, and it is supplied with batches of inputs of size 10,000 with a learning rate of 0.05 and a dropout probability of 0.0002. After running the neural network on 500 epochs with an adam optimizer, we get a 99.97% accuracy, which happens to be the best out of the several parameter tuning made to the network. Our predictive model using the SNN class of neural networks with a 99.97% accuracy appears to perform better than the KNN and SVM models, where the accuracy is 99.32% and 99.85%, respectively. We also look at the precision of our models in classifying the connections, and our SNN model appears to be a good predictor for Normal connections, DoS, R2L, and Probe attacks with 100%, 100%, 94%, and 99% precision, respectively.

Our self-normalizing neural network also appears to have better accuracy than Liu and Zhang [14], which uses a convolutional neural network with an accuracy of 98.02%. It is also performing better than Adams et al. [1], which used an artificial neural network with the hyperbolic tangent activation function, had an accuracy of 99.1%. The SNN also appears to perform better than

Pooja Purohit [21], which used a Bi-directional LSTM and performs better than most traditional machine learning methods, with an accuracy of 99.73%.

A possible future development is that this class of self-normalizing neural network can be evaluated against other well-known cyberattack datasets such as the NSL-KDD, UNSW-NB15, and ISCX 2012 [8]. These datasets have been generated to simulate more recent network traffic scenarios, wide varieties of low-footprint intrusion, and the possible attack complexities that may occur on a network in the present time, which are not present in the KDD Cup 99 dataset as it was generated two decades ago. The KDD Cup 99 dataset also suffers from certain limitations, such as redundancy of data, as a result of which bias may occur [20].

REFERENCES

1. Adams, S. O., Azikwe, E., & Zubair, M. A. (2022). Artificial neural network analysis of some selected KDD cup 99 dataset for intrusion detection. *Acta Informatica Malaysia*, *6*(2), 50-56.

2. Al-Mamory, S., & Jassim, F. (2013). Evaluation of different data mining algorithms with KDD cup 99 dataset. *Journal of Babylon University/Pure and Applied Sciences*, *21*(8), 2663-2681.

3. AL-Shabi, M. (2021). Design of a network intrusion detection system using complex deep neuronal networks. *International Journal of Communication Networks and Information Security*, *13*(3), 409-415.

4. Bebeshkoa, B., Khorolskaa, K., Kotenkoa N., Kharchenkoa, O., & Zhyrova, T. (2021). Use of neural networks for predicting cyberattacks. *CEUR Workshop Proceedings*, *2923*, 213-223.

5. Conover, W. J. (1999). *Practical nonparametric statistics*. Third edition. John Wiley Sons, Inc.

6. Dharamkar, B., & Singh R. (2014). A Review of cyber attack classification technique based on data mining and neural network approach. *International Journal of Computer Trends and Technology*, *7*(2), 100-105.

7. Fu, Y., Du, Y., Cao, Z., Li, Q., & Xiang, W. (2022). A deep learning model for network intrusion detection with imbalanced data. *Electronics*, *11*(6), 898. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics11060898.

8. Ghanem, W., & Jantan, A. (2019). Training a neural network for cyberattack classification applications using hybridization of an artificial bee colony and monarch butterfly optimization. *Neural Processing Letters*, *51*, 905-946. https://doi.org/10.1007/s11063019-10120-x

9. Hassan, M., Gumaei, A., Alsanada, A., Alrubaian, M., & Fortinoc, G. (2020). A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences: an International Journal*, *513*(C), 386-396. https://doi.org/10.1016/j.ins.2019.10.069.

10. Hoque, M. S., Mukit, M., Bikas, M., & Naser, A. (2012). An implementation of an intrusion detection system using a genetic algorithm. *International Journal of Network Security Its Applications*, *4*(2), 109-120. doi : 10.5121/ijnsa.2012.4208

11. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning with applications in R*. Second Edition. Springer.

12. Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. https://doi.org/10.48550/arXiv.1706.02515

13. Lantz, B. (2013). *Machine Learning with R.* Packt Publishing.

14. Liu, G., & Zhang, J. (2020). CNID: Research of network intrusion detection based on convolutional neural network. *Discrete Dynamics in Nature and Society*, 2020. https://doi.org/10.1155/2020/4705982

15. Maithem, M., & Al-sultany, G. A. (2021). Network intrusion detection system using deep neural networks. *Journal of Physics: Conference Series*, *1804*(1), 12138. https://doi.org/10.1088/1742-6596/1804/1/012138.

16. Mitchell, R., & Chen, I. (2013). A survey of intrusion detection techniques for cyber physical systems. *ACM Computing Survey*, *46*(4), 1-29. DOI:https://doi.org/10.1145/2542049

17. Moradi, M., & Zulkernine, M. (2004). A neural network based system for intrusion detection and classification of attacks. *IEEE International Conference on Advances in Intelligent Systems*, 148-04, https://www.researchgate.net/publication/260187937

18. Ogundokun, R., Awotunde, J., Sadiku, P., Adeniyi, E., Abiodun, M., & Dauda, O. (2021). An enhanced intrusion detection system using particle swarm optimization feature extraction technique. *Procedia Computer Science*, *193*,504-512. https://doi.org/10.1016/j.procs.2021.10.052.

19. Podder, P., Bharati S., M. Rubaiyat, Paul P., & Kose, U. (2021). Artificial neural network for cybersecurity: A comprehensive review. *Journal of Information Assurance and Security*, *16*(1), 10-23. https://doi.org/10.48550/arXiv.2107.01185

20. Priya, S., & Kumar, K. (2021). Performance analysis comparison on various cyber-attack dataset by relating a deep belief network model on an intrusion detection system. *Information Technology In Industry*, *9*(3), 608-613.

21. TS, P., & Shrinivasacharya, P. (2021). Evaluating neural networks using bi-directional LSTM for network IDS (intrusion detection systems) in cyber security. *Global Transitions Proceedings*, 2(2), 448-454. https://doi.org/10.1016/j.gltp.2021.08.017.

22. Weidman, S. (2019). *Deep learning from scratch.* O'Reilly Media, Inc.

APPENDIX A

LETTER FROM INSTITUTIONAL RESEARCH BOARD

MARSHALL UNIVERSITY
www.marshall.edu

Office of Research Integrity

September 27, 2022

Oluwapelumi Eniodunmo
Department of Mathematics
Smith Hall 523
Marshall University

Dear Oluwapelumi:

This letter is in response to the submitted thesis abstract entitled *"A Predictive Model to Predict Cyberattack Using Self Normalizing Neural Networks."* After assessing the abstract, it has been deemed not to be human subject research and therefore exempt from oversight of the Marshall University Institutional Review Board (IRB). The Code of Federal Regulations (45CFR46) has set forth the criteria utilized in making t/his determination. Since the information in this study does not involve human subjects as defined in the above referenced instruction, it is not considered human subject research. If there are any changes to the abstract, you provided then you would need to resubmit that information to the Office of Research Integrity for review and a determination.

I appreciate your willingness to submit the abstract for determination. Please feel free to contact the Office of Research Integrity if you have any questions regarding future protocols that may require IRB review.

Sincerely,

Bruce F. Day, ThD, CIP
Director

APPENDIX B

CODES

Feature Selection for Categorical Variables

```
# Categorical feature selection

from numpy import unique
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
import pandas
import numpy

number_of_features_to_select = 9

missing_values = ["n/a", "na", "--"]
df = pandas.read_csv("kdd_cyberattack.csv", na_values=missing_values, usecols=['
    protocoltype', 'service', 'flag', 'land', 'loggedin', 'rootshell', 'suattempted', '
    ishostlogin', 'isguestlogin', 'labels'])
mapping = {
    'back.': 'DOS', 'land.': 'DOS', 'neptune.': 'DOS', 'pod.': 'DOS', 'smurf.': 'DOS', '
        teardrop.': 'DOS',
    'ftp_write.': 'R2L', 'guess_passwd.': 'R2L', 'imap.': 'R2L', 'multihop.': 'R2L', 'phf
        .': 'R2L', 'spy.': 'R2L',
    'warezclient.': 'R2L', 'warezmaster.': 'R2L',
    'buffer_overflow.': 'U2R', 'loadmodule.': 'U2R', 'perl.': 'U2R', 'rootkit.': 'U2R',
    'ipsweep.': 'Probe', 'nmap.': 'Probe', 'portsweep.': 'Probe', 'satan.': 'Probe',
    'normal.': 'Normal',
}
df['labels'] = df.labels.map(mapping)
print(df.columns)
columns = df.columns.values
array = df.values
X = array[:, 0:len(array[0]) - 1]
print(X[0:5])

Y = array[:, len(array[0]) - 1]
print(Y[0:5])
print(unique(Y))

oe = OrdinalEncoder()
oe.fit(X)
X = oe.transform(X)
le = LabelEncoder()
le.fit(Y)
Y = le.transform(Y)
fs = SelectKBest(score_func=mutual_info_classif, k=number_of_features_to_select)
fs.fit(X, Y)
scores = fs.scores_
```

```
sortedScore = numpy.sort(scores)[::-1]
print(scores)
print(sortedScore)
bag = {}
for index, value in numpy.ndenumerate(scores):
    if numpy.isnan(value):
        continue
    bag[value] = index
cols = []

loop = 0
feature_count = 0
print(len(sortedScore))
while feature_count < number_of_features_to_select and loop < len(sortedScore):
    if numpy.isnan(sortedScore[loop]):
        print("loop is ", loop, "and sorted score is ", sortedScore[loop])
        loop += 1
    else:
        print("loop is ", loop, "and sorted score is ", sortedScore[loop])
        cols.append(columns[bag[sortedScore[loop]]])
        feature_count += 1
        loop += 1


print(cols)
```

Feature Selection for Numeric Variables

```
# Numeric feature selection using

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
import pandas
import numpy

number_of_features_to_select = 32


missing_values = ["n/a", "na", "--"]
df = pandas.read_csv("kdd_cyberattack.csv", na_values=missing_values, usecols=['duration
    ', 'srcbytes', 'dstbytes', 'wrongfragment',
       'urgent', 'hot', 'numfailedlogins',
       'numcompromised', 'numroot',
       'numfilecreations', 'numshells', 'numaccessfiles', 'numoutboundcmds',
       'count', 'srvcount', 'serrorrate',
       'srvserrorrate', 'rerrorrate', 'srvrerrorrate', 'samesrvrate',
       'diffsrvrate', 'srvdiffhostrate', 'dsthostcount', 'dsthostsrvcount',
       'dsthostsamesrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate',
       'dsthostsrvdiffhostrate', 'dsthostserrorrate', 'dsthostsrvserrorrate',
       'dsthostrerrorrate', 'dsthostsrvrerror_rate', 'labels'])
mapping = {
    'back.': 'DOS', 'land.': 'DOS', 'neptune.': 'DOS', 'pod.': 'DOS', 'smurf.':'DOS', '
```

```
        teardrop.': 'DOS',
    'ftp_write.': 'R2L', 'guess_passwd.': 'R2L', 'imap.': 'R2L', 'multihop.': 'R2L', 'phf
        .': 'R2L', 'spy.': 'R2L', 'warezclient.': 'R2L', 'warezmaster.': 'R2L',
    'buffer_overflow.': 'U2R', 'loadmodule.': 'U2R', 'perl.': 'U2R', 'rootkit.': 'U2R',
    'ipsweep.': 'Probe', 'nmap.': 'Probe', 'portsweep.': 'Probe', 'satan.': 'Probe',
    'normal.': 'Normal',
}
df['labels'] = df.labels.map(mapping)
print(df.columns)
columns = df.columns.values
array = df.values
X = array[:,0:len(array[0])-1]
Y = array[:,len(array[0])-1]

test = SelectKBest(score_func=f_classif, k=number_of_features_to_select)
fit = test.fit(X, Y)
scores = fit.scores_
sortedScore = numpy.sort(scores)[::-1]
print(scores)

bag = {}
for index, value in numpy.ndenumerate(scores):
    if numpy.isnan(value):
        continue
    bag[value] = index
cols = []

loop=0; feature_count=0
print(len(sortedScore))
while feature_count < number_of_features_to_select and loop<len(sortedScore):
    if numpy.isnan(sortedScore[loop]):
        print("loop is ", loop, "and sorted score is ", sortedScore[loop])
        loop+=1
    else:
        print("loop is ", loop, "and sorted score is ", sortedScore[loop])
        cols.append(columns[bag[sortedScore[loop]]])
        feature_count+=1
        loop+=1

print(cols)
```

Self Normalizing Neural Network (SNN)

```
import math
from collections import OrderedDict

import matplotlib.pyplot as plt
import numpy as np
import time

import pandas as pd
import seaborn as sns
import torch as T
```

```
from numpy import unique
from sklearn.metrics import confusion_matrix, classification_report,
    plot_confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, StandardScaler

device = T.device("cpu") # apply to Tensor or Module

selected_categorical_non_binary_columns = ['service', 'protocoltype', 'flag']
selected_categorical_binary_columns = ['loggedin', 'isguestlogin', 'rootshell']
selected_numeric_columns = ['count', 'dsthostcount', 'srvcount', 'dsthostdiffsrvrate', '
    dsthostsamesrcportrate',
                        'dsthostsrvdiffhostrate', 'hot', 'diffsrvrate', 'srvdiffhostrate
                            ', 'samesrvrate',
                        'dsthostsrvserrorrate', 'serrorrate', 'srvserrorrate', '
                            dsthostserrorrate', 'rerrorrate',
                        'dsthostsrvrerror_rate', 'srvrerrorrate', 'dsthostrerrorrate']
target = ['labels']

columns_to_use = selected_numeric_columns + selected_categorical_binary_columns +
    selected_categorical_non_binary_columns + target
print("columns to use is ", columns_to_use)


class Dataset(T.utils.data.Dataset):

    def __init__(self, src_file, n_rows=None):
        all_xy = pd.read_csv(src_file, usecols=columns_to_use)
        mapping = {
            'back.': 'DOS', 'land.': 'DOS', 'neptune.': 'DOS', 'pod.': 'DOS', 'smurf.': '
                DOS', 'teardrop.': 'DOS',
            'ftp_write.': 'R2L', 'guess_passwd.': 'R2L', 'imap.': 'R2L', 'multihop.': 'R2L
                ', 'phf.': 'R2L',
            'spy.': 'R2L',
            'warezclient.': 'R2L', 'warezmaster.': 'R2L',
            'buffer_overflow.': 'U2R', 'loadmodule.': 'U2R', 'perl.': 'U2R', 'rootkit.': '
                U2R',
            'ipsweep.': 'Probe', 'nmap.': 'Probe', 'portsweep.': 'Probe', 'satan.': 'Probe
                ',
            'normal.': 'Normal',
        }
        all_xy['labels'] = all_xy.labels.map(mapping)
        print("total number of columns is ", len(all_xy.columns))

        n = len(all_xy)
        print("total number of record is ", n)
        tmp_x = all_xy.iloc[0:n, 0:len(all_xy.columns) - 1]
        tmp_y = all_xy.iloc[0:n, len(all_xy.columns) - 1]

        oe = OrdinalEncoder()
        tmp_x[selected_categorical_non_binary_columns] = oe.fit_transform(
            tmp_x[selected_categorical_non_binary_columns])
```

```
        scaler = StandardScaler()
        tmp_x[selected_numeric_columns] = scaler.fit_transform(tmp_x[
            selected_numeric_columns])

        global le
        le = LabelEncoder()
        tmp_y = le.fit_transform(tmp_y[:, ])

        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(tmp_x,
            tmp_y, test_size=0.20,
                                                                                 random_state
                                                                                     =12345)

        self.x_data = T.tensor(self.X_train.values, dtype=T.float32).to(device)
        self.y_data = T.tensor(self.y_train, dtype=T.int64).to(device)

    def __len__(self):
        return len(self.x_data)

    def __getitem__(self, idx):
        preds = self.x_data[idx]
        trgts = self.y_data[idx]
        sample = {
            'predictors': preds,
            'targets': trgts
        }
        return sample

    # -----------------------------------------------------------


class Net(T.nn.Module):
    def __init__(self, in_dim, n_layers=5, hidden_dim=32, dropout_probability=0.8):
        super().__init__()
        n_layers = n_layers
        hidden_dim = hidden_dim
        out_dim = 5
        dropout_prob = dropout_probability

        print("in neural network")
        print("input dim is ", in_dim)
        print("n layers is ", n_layers)
        print("hidden dim is ", hidden_dim)
        print("output dim is ", out_dim)
        print("dropout prob is ", dropout_prob)

        layers = OrderedDict()
        for i in range(n_layers - 2):
            if i == 0:
                layers[f"fc{i}"] = T.nn.Linear(in_dim, hidden_dim, bias=False)
            else:
```

```
                layers[f"fc{i}"] = T.nn.Linear(hidden_dim, hidden_dim, bias=False)
            layers[f"selu_{i}"] = T.nn.SELU()
            layers[f"dropout_{i}"] = T.nn.AlphaDropout(p=dropout_prob)

        layers[f"fc{i + 1}"] = T.nn.Linear(hidden_dim, 5, bias=False)
        layers[f"selu_{i + 1}"] = T.nn.SELU()
        layers[f"dropout_{i + 1}"] = T.nn.AlphaDropout(p=dropout_prob)

        layers[f"fc_{i + 2}"] = T.nn.Linear(out_dim, out_dim, bias=True)
        self.network = T.nn.Sequential(layers)
        self.reset_parameters()

    def reset_parameters(self):
        for layer in self.network:
            if not isinstance(layer, T.nn.Linear):
                continue
            T.nn.init.normal_(layer.weight, std=1 / math.sqrt(layer.out_features))
            if layer.bias is not None:
                fan_in, _ = T.nn.init._calculate_fan_in_and_fan_out(layer.weight)
                bound = 1 / math.sqrt(fan_in)
                T.nn.init.uniform_(layer.bias, -bound, bound)

    def forward(self, x):
        return self.network(x)

    # -----------------------------------------------------------


def accuracy(model, ds):
    # assumes model.eval()
    # granular but slow approach
    n_correct = 0
    n_wrong = 0
    for i in range(len(ds)):
        X = ds[i]['predictors']
        Y = ds[i]['targets'] # [0] [1] or [2]
        with T.no_grad():
            oupt = model(X) # logits form

        big_idx = T.argmax(oupt) # [0] [1] or [2]
        if big_idx == Y:
            n_correct += 1
        else:
            n_wrong += 1

    acc = (n_correct * 1.0) / (n_correct + n_wrong)
    return acc


# -----------------------------------------------------------------

def accuracy_quick(model, dataset):
```

```
    # assumes model.eval()
    # en masse but quick
    n = len(dataset)
    X = dataset[0:n]['predictors']
    Y = T.flatten(dataset[0:n]['targets']) # 1-D

    with T.no_grad():
        oupt = model(X)
        # (_, arg_maxs) = T.max(oupt, dim=1) # old style
    arg_maxs = T.argmax(oupt, dim=1) # collapse cols
    num_correct = T.sum(Y == arg_maxs)
    acc = (num_correct * 1.0 / len(dataset))
    return acc.item()
```

```
def accuracy2(y_true, y_pred):
    correct_predictions = 0
    # iterate over each label and check
    for true, predicted in zip(y_true, y_pred):
        if true == predicted:
            correct_predictions += 1
    # compute the accuracy
    accuracy = correct_predictions / len(y_true)
    return accuracy
```

```
# -----------------------------------------------------------

def getTrainDataSet():
    train_file = "../../../data/kdd_cyberattack.csv"
    train_ds = Dataset(train_file)
    return train_ds, le


def main(batch_size=20000, max_epoch=500, lr_rate=0.01, n_layers=5, hidden_dim=32,
    dropout_probability=0.8, optim='sgd'):
    # 0. get started
    np.random.seed(1)
    T.manual_seed(1)

    # 1. create DataLoader objects
```

```
print("Getting cyberattack dataset ")


train_ds, le = getTrainDataSet()

bat_size = batch_size
train_ldr = T.utils.data.DataLoader(train_ds, batch_size=bat_size, shuffle=True)

# 2. create network
net = Net(train_ds.X_train.shape[1], n_layers=n_layers, hidden_dim=hidden_dim,
    dropout_probability=dropout_probability).to(device)

# 3. train model
max_epochs = max_epoch
ep_log_interval = 100
lrn_rate = lr_rate

# ------------------------------------------------------------

loss_func = T.nn.CrossEntropyLoss() # apply log-softmax()
if optim == 'adam':
    optimizer = T.optim.Adamax(net.parameters(), lr=lrn_rate)
else:
    optimizer = T.optim.SGD(net.parameters(), lr=lrn_rate)

print("\nbat_size = %3d " % bat_size)
print("loss = " + str(loss_func))
print("optimizer = {0}".format('adam' if optim=='adam' else 'SGD'))
print("max_epochs = %3d " % max_epochs)
print("lrn_rate = %0.3f " % lrn_rate)

print("\nStarting train with saved checkpoints")
net.train()
for epoch in range(0, max_epochs):
    T.manual_seed(1 + epoch) # recovery reproducibility
    epoch_loss = 0 # for one full epoch
    for (batch_idx, batch) in enumerate(train_ldr):
        X = batch['predictors'] # inputs
        Y = batch['targets'] # shape [10,3] (!)

        optimizer.zero_grad()
        oupt = net(X) # shape [10] (!)

        loss_val = loss_func(oupt, Y) # avg loss in batch
        epoch_loss += loss_val.item() # a sum of averages
        loss_val.backward()
        optimizer.step()

    if epoch % ep_log_interval == 0:
        print("epoch = %4d loss = %0.4f" % \
              (epoch, epoch_loss))
```

```
    print("Training complete ")

    # 4. evaluate model accuracy
    print("\nComputing model accuracy")
    net.eval()
    acc_train = accuracy(net, train_ds) # item-by-item
    print("Accuracy on training data = %0.4f" % acc_train)
    acc_train = accuracy_quick(net, train_ds) # item-by-item
    print("Quick Accuracy on training data = %0.4f" % acc_train)


    inpt = T.tensor(train_ds.X_test.values, dtype=T.float32).to(device)
    with T.no_grad():
        logits = net(inpt) # values do not sum to 1.0
    probs = T.softmax(logits, dim=1) # tensor


    _, y_pred_tags = T.max(probs, dim=1)


    train_ds.y_test = le.inverse_transform(train_ds.y_test)
    y_pred_tags = le.inverse_transform(y_pred_tags)



    report = classification_report(train_ds.y_test, y_pred_tags)
    conf_matrix = confusion_matrix(train_ds.y_test, y_pred_tags)
    print(report)
    print(conf_matrix)

    ConfusionMatrixDisplay.from_predictions(train_ds.y_test, y_pred_tags)
    plt.savefig('foo_{0}_{1}_{2}_{3}_{4}_{5}.png'.format(batch_size, max_epoch, lr_rate,
        n_layers, hidden_dim, dropout_probability), bbox_inches='tight')


if __name__ == "__main__":

    main(batch_size=10000, max_epoch=800, lr_rate=0.01, n_layers=5, hidden_dim=30,
        dropout_probability=0.0, optim='adam')
```

k-Nearest Neighbor(kNN)

```
import math

from sklearn.neighbors import KNeighborsClassifier

from results.selfnormresults.result2.result2 import getTrainDataSet
from results.selfnormresults.result1.result1 import accuracy2
from matplotlib import pyplot as plt
from numpy import unique
from sklearn.metrics import classification_report, confusion_matrix,
    ConfusionMatrixDisplay
```

```
train_ds, le = getTrainDataSet()



k = int(math.sqrt(len(train_ds.X_train)))
k = k if k % 2 != 0 else k + 1
classifier = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
classifier.fit(train_ds.X_train, train_ds.y_train)
y_pred = classifier.predict(train_ds.X_test)



train_ds.y_test = le.inverse_transform(train_ds.y_test)
y_pred = le.inverse_transform(y_pred)


report = classification_report(train_ds.y_test, y_pred)
conf_matrix = confusion_matrix(train_ds.y_test, y_pred)
print(report)
print(conf_matrix)
acc_train = accuracy2(train_ds.y_test, y_pred) # item-by-item
print("Accuracy on training data = %0.4f" % acc_train)

ConfusionMatrixDisplay.from_predictions(train_ds.y_test, y_pred)
plt.savefig('foo_{0}.png'.format(k), bbox_inches='tight')
```

Support Vector Machine (SVM)

```
from matplotlib import pyplot as plt
from numpy import unique
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix,
    ConfusionMatrixDisplay


from results.selfnormresults.result1.result1 import getTrainDataSet
from results.selfnormresults.result1.result1 import accuracy2


def runResult(classifier, train_ds, le, C, kernel, param=""):
    y_pred = classifier.predict(train_ds.X_test)

    train_ds.y_test = le.inverse_transform(train_ds.y_test)
    y_pred = le.inverse_transform(y_pred)

    report = classification_report(train_ds.y_test, y_pred)
    conf_matrix = confusion_matrix(train_ds.y_test, y_pred)
    print(report)
    print(conf_matrix)
    acc_train = accuracy2(train_ds.y_test, y_pred) # item-by-item
    print("Accuracy on training data = %0.4f" % acc_train)
```

```
    ConfusionMatrixDisplay.from_predictions(train_ds.y_test, y_pred)
    plt.savefig('foo_{0}_{1}_{2}.png'.format(C, kernel, param), bbox_inches='tight')
```

```
train_ds, le = getTrainDataSet()
C = 0.1
kernel='linear'
param = ''
classifier = svm.SVC(C=C, kernel=kernel).fit(train_ds.X_train, train_ds.y_train)
runResult(classifier, train_ds, le, C, kernel, param)
```