A MACHINE LEARNING APPROACH FOR UNIFORM INTRUSION DETECTION

by

Saurabh Devulapalli

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Thomas J. Hacker, Chair

Department of Computer and Information Technology

Dr. John A. Springer

Department of Computer and Information Technology

Dr. Baijian Yang

Department of Computer and Information Technology

Approved by:

Dr. John A. Springer

ACKNOWLEDGMENTS

I would like to thank my major advisor, Dr. Thomas Hacker, for his guidance, encouragement, and supervision as well as for teaching me to be persistent in my research.

I would also like to thank the Graduate Committee members, Dr. John Springer and Dr. Baijian Yang for serving on my graduate committee.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ML	Machine Learning
DL	Deep Learning
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
DoS	Denial of Service
R2L	Remote to Local
U2R	User to Root
SVM	Support Vector Machine
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
SMOTE	Synthetic Minority Oversampling Technique

GLOSSARY

Intrusion detection – Intrusion detection is the process of safeguarding computers and data networks from attacks and misuse. (Mukherjee, Heberlein, & Levitt, 1994)

Machine Learning – A branch of computer science that deals with learning statistical and probabilistic distribution of data and improving with data augmentation without explicit programming (Shalev-Shwartz & Ben-David, 2014)

Ensemble learning – It is the process of combining multiple machine learning models to achieve better predictive performance than that achieved by any of the individual models. (Dietterich et al., 2002)

Deep Learning – "A branch of machine learning involving the use of artificial neural networks to automatically learn features of data." (Goodfellow, Bengio, Courville, & Bengio, 2016)

Training – The process of providing a Machine Learning model with data to learn from.

Testing – The process of evaluating the performance of the Machine Learning algorithm developed.

ABSTRACT

Intrusion Detection Systems are vital for computer networks as they protect against attacks that lead to privacy breaches and data leaks. Over the years, researchers have formulated intrusion detection systems (IDS) using machine learning and/or deep learning to detect network anomalies and identify four main attacks namely, Denial of Service (DoS), Probe, Remote to Local (R2L) and User to Root (U2R). However, the existing models are efficient in detecting just few of the aforementioned attacks while having inadequate detection rates for the rest. This deficiency makes it difficult to choose an appropriate IDS model when a user does not know what attacks toexpect. Thus, there is a need for an IDS model that can detect, with uniform efficiency, all the four main classes of network intrusions. This research is aimed at exploring a machine learning approach to an intrusion detection model that can detect DoS, Probe, R2L and U2R attack classeswith uniform and high efficiency. A mulitlayer perceptron was trained in an ensemble with J48 decision tree. The resultant ensemble learning model achieved over 85% detection rates for each of DoS, probe, R2L, and U2R attacks.

CHAPTER 1. INTRODUCTION

This chapter talks about the problem addressed in this research. It gives a background to the problem and talks about its significance. It also discusses the scope of the research.

1.1 Background

In the computer security world, the term intrusion refers to any unauthorized access to an information system. Intrusion Detection Systems (IDS) are technologies that identify unauthorized access and suspicious activity and policy violations in computer systems and networks (Rowland, 2002). They can be in the form of devices or software applications and are meant for preserving confidentiality, integrity, and availability of computer systems (Forouzan, 2007). Network Intrusion Detection Systems (NIDS) are intrusion detection systems that monitornetwork traffic for malicious activity and attacks. According to Cole (2011) there are four main classes of network attacks:

1. Denial of Service (DoS): In this type of attack, computer users are unable to access a computer network because the attacker has flooded it with traffic.
2. Probe Attacks: This is a type of attack where the hacker surveys a computer network for its vulnerabilities.
3. Remote to Local (R2L): In this type of attack, the hacker gains unauthorized access to a machine on a network he/she is not a part of.
4. User to Root (U2R): In this kind of attack, the hacker attempts to illegally gain administrative privileges to a computer.

The task of an IDS is generally to classify network traffic behaviour as normal or abnormal.

1.1.1 Machine Learning and Deep Learning in Intrusion Detection

Machine Learning (ML) is one of the branches of artificial intelligence that deals with programming computers to improve automatically by learning from past experience (Bishop, 2006). Deep learning (DL) is a field in ML with a similar goal but deals with artificial neural networks for learning from past experience (Goodfellow et al., 2016). In the past two decades, these two branches of artificial intelligence have helped IDS become smarter and more threat-aware, improving the security of networks significantly. Researchers incorporated ML andDL into IDS so that they learn attack features from known attacks and identify incoming unseen attacks. Machine learning and deep learning tasks can be categorized as classification or regression. Classification is the process of categorizing data based on what is learned from previous data while regression is the process of predicting the value of an unseen data sample based on what on learned from previous data. In the context of intrusion detection, machine learning and deep learning based IDS learn network connection features from existing data and perform classification tasks on unseen network traffic by categorizing them as normal or abnormal. These tasks can be considered as two-class problems - normal or abnormal or a multi-class problems - normal, DoS, probe, R2L, or U2R.

1.2 Problem

In the real world, computer users cannot predict which class of attacks out of DoS, Probe, R2L, and U2R they can expect. Even though one class of attacks may be more common than the others, it is wise to be prepared for all kinds of attacks. Although the existing ML and DL based IDS achieve high average classification accuracy for DoS, probe, R2L, and U2R classes of attacks, the difference among the classification accuracy figures for individual attack classes is considerable. For instance, if the IDS works well in detecting DoS, probe and R2L attacks, it is less efficient in detecting U2R attacks (Yang, Zheng, Wu, Niu, & Yang, 2019). With regard to security of computer networks, such an IDS might not be very effective when the majority of attacks the network sees in the future are U2R attacks. This poses a problem for the computer users or network security administrators in selecting an appropriate IDS model, especially when

the user does not know what attacks to anticipate. This deficiency in capability of the existing IDS creates a need for a machine learning or deep learning based IDS that works uniformly across all attack classes so that network security is improved.

1.3 Significance

With the expansion of the internet and computer networks, security has become a crucial aspect of computer networks. According to a blog written by the East Coast Polytechnic University, no computer network is invulnerable to attacks and a solid network security system is required to protect it from malicious entities (*"Why Do We Need Network Security?"*, n.d.). An Intrusion Detection System is one such network security system that monitors computer systems for cyber attacks. Intrusion detection systems identify and report malicious behaviour in computer networks so that network administrators can activate defense protocols in a timely manner to mitigate the attack. Over the years, cyber attacks have become more and more sophisticated and will only continue to become more complex. Therefore, the modern world IDS needs to be adaptable and learn from previous attacks so that new and unseen attacks can be automatically detected. For this purpose, researchers incorporated machine learning into IDS so that they automatically learn from known past data and detect new and unfamiliar attacks by recognizing attack patterns.

1.4 Purpose

The purpose of this study is to explore machine learning approaches to build an artificial intelligence based Intrusion Detection System that is capable of automatically detecting four main classes of network intrusions, i.e., Denial of Service, Probe, User to Root, and Remote to Local, with uniform efficiency.

1.5 Research Questions

The questions addressed by this study are:

1. Is it possible to have a artificial intelligence based IDS model that can classify all the four kinds of network intrusions with uniform efficiency?

2. Would there be a trade-off between achieving uniform efficiency across all attack classes and achieving high efficiency for individual classes?

1.6 Assumptions

This research uses a benchmark dataset for intrusion detection called NSL-KDD (*NSL-KDD dataset*, n.d.) to evaluate the proposed IDS model. The underlying assumption is thatthe results obtained on this benchmark dataset are applicable to the real world.

1.7 Limitations

This research's limitation is that the proposed intrusion detection model needs all or most of the data on network connections to perform efficiently. Incomplete or missing information on connections may lead to misclassification of attacks. Also, the speed of the proposed model depends upon the computational power of the underlying hardware (Graphics Processing Unit). Inefficient hardware can slow down the proposed model.

1.8 Delimitations

The study's scope is limited to the four main classes of network intrusions, i.e., Denial of Service (DoS), Probing, User to Remote (U2R), and Remote to Local (R2L) attacks. The proposed model will be trained and tested on these four classes of attacks only and the findings ofthis research are not be applicable to other classes of attacks.

1.9 Summary

This chapter introduced the concepts of intrusion detection system, machine learning, and deep learning. It provided a background to the research problem and summarized the purpose, research questions, assumptions, and limitations involved in this study. In addition to that, the delimitations were also discussed to provide the reader an understanding of the scope of this research.

CHAPTER 2. REVIEW OF LITERATURE

This chapter involves a discussion of the literature relevant to the problem of intrusion detection systems that are based on machine learning and deep learning techniques.

2.1 Related Work

This section summarizes the previous research related to machine learning and deep learning based IDS. Mishra et al. (2018) provided a detailed survey of a number of ML based intrusion detection systems. Liu and Lang (2019) surveyed intrusion detection models that were based on ML and also deep learning. Gamage and Samarabandu (2020) presented a comprehensive survey of previous surveys of ML and DL based IDS. Based on the findings in these three surveys, the intrusion detection models that have been presented so far by previous researchers can be categorized into three main divisions – Individual Machine Learning models, Ensemble models and Deep Learning models.

2.1.1 Individual Machine Learning Models

Kim and Park (2003) introduced a Support Vector Machine (SVM) (Cortes & Vapnik, 1995) based IDS that treated the intrusion detection problem as a two-class (normal/attack) classification task as well as a multiclass (normal/DoS/Probe/U2R/R2L) classification task. Kim'smodel was trained on KDD Cup 99 dataset (Tavallaee et al., 2009) and while it was good at classifying probe and DoS attacks with accuracies of 83.3% and 97.1% respectively, the classification accuracies for U2R and R2L attacks were just 13.2% and 8.4% due to insufficient training samples in the dataset. Amor, Benferhat, and Elouedi (2004) also worked on the same KDD Cup 99 dataset but proposed an approach to network intrusion detection that was based on Naive Bayes algorithm. Amor's results, however, were very similar to Kim's in the sense that theclassification accuracy was poor for classes with few training samples (U2R and R2L). Lee, Lee,Sohn, Ryu, and Chung (2008) evaluated the performance of decision trees in intrusion detection and found out that decision trees performed unsatisfactorily in detecting network intrusions, achieving classification accuracies of just 75.5%, 82.6%, 58.6% and 78.1% on DoS, Probe, U2R and R2L respectively. Stein, Chen, Wu,

and Hua (2005) also used decision trees for intrusion detection but by employing Genetic Algorithms (Mitchell, 1998) based feature selection. This inclusion of genetic algorithms made feature selection more efficient as they determined what features are used in creating the decision tree, thereby producing an optimized tree. Interestingly,the combined model achieved a very high overall classification accuracy on KDD Cup 99 dataset.The accuracy figures reached 97.8%, 99.1% and 99.9% for DoS, Probe and U2R attacks respectively but could not go higher than 80.4% for R2L attacks. Hasan, Nasser, Pal, and Ahmad(2014) tried to solve the intrusion detection problem using Random Forests on the KDD Cup 99 dataset and achieved classification accuracy of 98.9% for DoS, 100% for U2R but only 55.1% forProbe and 66.7% for R2L. Lin, Ke, and Tsai (2015) used an unsupervised learning approach (Barlow, 1989) to intrusion detection by empying k-nearest neighbours algorithm (Keller, Gray, & Givens, 1985). Lin's approach could not yield satisfactory results as the classification accuracyfor R2L and U2R attacks was just 57% and 3.9% respectively. Once again, the proposed model failed on at least one of the classes of network intrusions. Based on the findings above, and especially by comparing the works of Lee et al. (2008) and Stein et al. (2005), it can be concludedthat traditional ML models are not very effective in intrusion detection when they are used individually. This brings us to the next category – Ensemble methods.

2.1.2 Ensemble Models

Ensemble learning is the process by which two or more machine learning models are combined to produce a model with higher predictive power (Dietterich et al., 2002). Over the years, researchers have employed ensemble methods for intrusion detection to achieve better classification results. Mukkamala et al. (2005) proposed an ensemble of hard computing and soft computing paradigms (Zadeh, 1996) to efficiently detect network intrusions. Their model was a combination of a feedforward artificial neural network (Bebis & Georgiopoulos, 1994), SVM (Cortes & Vapnik, 1995) and multivariate adaptive regression splines (Friedman, 1991). Mukkamala evaluated each individual model as well as the ensemble on the KDD Cup 99 dataset and observed that the ensemble outperformed each of the component models. The ensemble model achieved classification accuracies of 99.8% on Probe, 99.9% on DoS, 76.0% on U2R and 100% on R2L attacks. These accuracy figures are much superior to those achieved by researchers

referenced in the previous section, who used machine learning algorithms individually. Aburomman and Reaz (2016) introduced another ensemble method that combined support vector machines, k-nearest neighbors (Keller et al., 1985) and particle swarm optimization (Kennedy et al., 1995). When trained and evaluated on the KDD Cup 99 dataset, Aburomman's model achieved 96.7% classification accuracy for Probe attacks, 98.8% for DoS, 99.8% for U2R and 84.8% for R2L. Once again, the ensemble methods proved to be better than individual ML algorithms for intrusion detection. However, in both the ensemble approaches discussed above, the drawback is that the accuracies are not uniform for all attack classes. For instance, Mukkamala's model is not as efficient for U2R as it is for the other classes while Aburomman's model is less efficient for R2L than it is for the other classes.

2.1.3 Deep Learning Models

In recent years, researchers have started using deep learning as a solution to the problem of intrusion detection. Yin et al. (2017) introduced a deep learning based intrusion detection model which employed recurrent neural networks (Gers, Schmidhuber, & Cummins, 1999) to extract feature representations from attack data and automatically learn attack patterns. Yin studied the relation between depth of the neural network and its performance and developed a model that has a very low overall false positive rate of 1.27%. Yin evaluated their model on the NSL-KDD dataset and achieved classification accuracies of 83.4% for Probe, 83.5% for DoS, 11.5% for U2Rand 24.7% for R2L attacks. Yang et al. (2019) proposed the use of deep belief networks (Hinton,Osindero, & Teh, 2006) for network intrusion detection. They achieved classification accuracies of 90.5%, 92.3%, 68.4% and 95.9% for probe, DoS, U2R and R2L classes of attacks respectively.While deep learning models seem to perform better than traditional ML models, once again, it canbe observed that the accuracy figures are not uniform for all the attack classes. Yang's model underperforms for R2L attacks while Yin's model underperforms for U2R and R2L attacks. Based on the above findings, it can be concluded that there is a need for an intrusion detectionmodel that achieves uniformly high classification accuracies for all attack classes.

Table 2.1. *Previous IDS models and their results*

Literature	Method	Dataset	Classification Accuracy (%)			
			DoS	Probe	R2L	U2R
Kim et al. (2003)	SVM	KDD Cup 99	97.1	83.3	8.4	13.2
Amor et al. (2004)	Naive Bayes	KDD Cup 99	96.4	78.2	7.1	11.8
Stein et al. (2005)	Decision Tree	KDD Cup 99	97.8	99.13	80.4	99.9
Lee et al. (2008)	Decision Tree	DARPA	82.6	75.5	78.1	58.6
Lin et al. (2015)	CANN	KDD Cup 99	99.7	87.6	57.0	3.9
Hasan et al. (2014)	Random Forest	KDD Cup 99	98.9	66.7	66.7	100
Mukkamala et al. (2005)	ANN + SVM + MARS	DARPA	99.9	99.8	100	76.0
Aburomman et al. (2016)	SVM + kNN + PSO	KDD Cup 99	98.8	96.7	84.8	99.8
Yin et al. (2017)	RNN	NSL-KDD	83.5	83.4	24.7	11.5
Potluri et al. (2018)	CNN	NSL-KDD	84.2	85.4	0	0
Yang et al. (2019)	DBN	NSL-KDD	92.3	90.5	95.9	68.4

2.2 Important Concepts Pertaining to the Proposed Approach

This section talks about some concepts that are needed to understand the proposed approach, which will be discussed in the subsequent section.

2.2.1 Ensemble Learning

This subsection explains ensemble learning and discusses the various kinds of ensemble methods that are used in practice. According to their book on hands-on machine learning, Géron (2019) explained ensemble learning as follows:

> ”Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases you will find that this aggregated answer is better than an expert’s answer. This is called the wisdom of the crowd. Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor. A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method.” (p. 189)

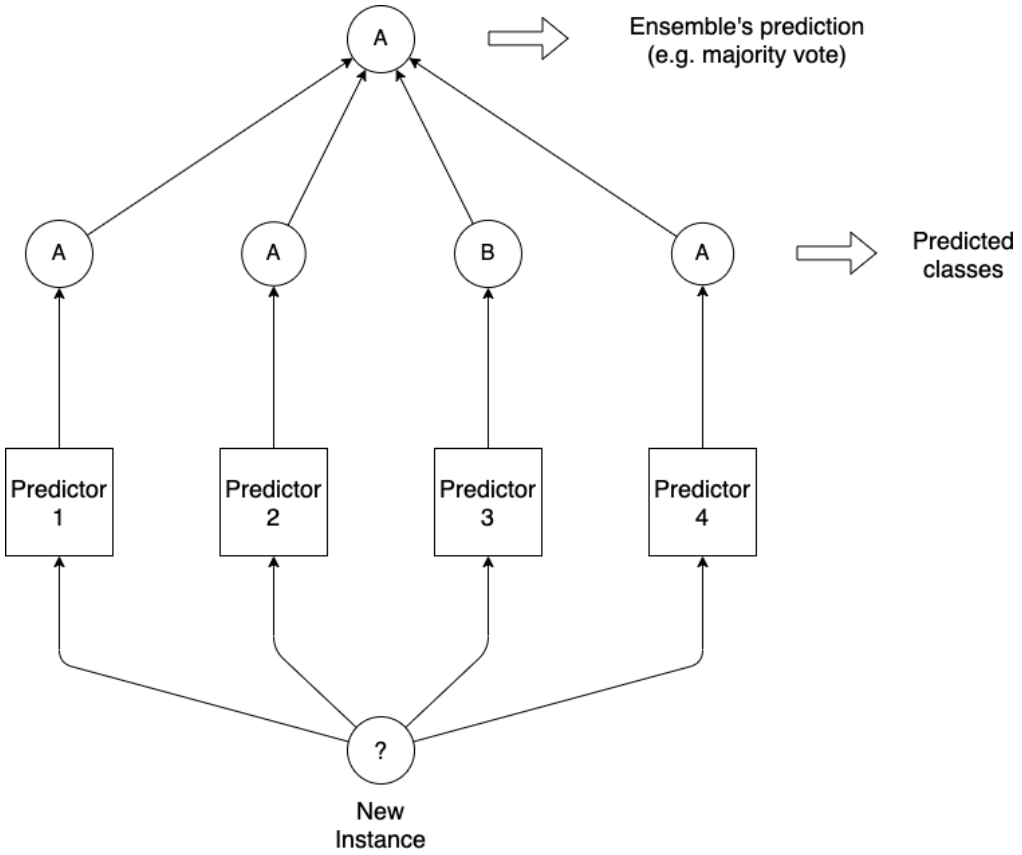

Figure 2.1. Ensemble model predicting the class with most votes.

In other words, ensemble learning is a technique which combines individual machine learning algorithms to achieve better prediction results than those achieved by the said individual algorithms. In the context of this research, prediction refers to the task of classification. The advantages of using ensemble methods over individual machine learning algorithms are: 1) as discussed above, their predictive power is better than that of their constituent algorithms, and 2) the variance in the predictions is reduced.

There are various kinds of ensemble methods but for the scope of this research, listed below are two of them:

- Voting Classifier: This type of ensemble method performs classification tasks based on voting. The class predictions from each of its constituent ML algorithms are aggregated and the class with the most predictions (votes) is deemed as the class predicted by the ensembles. This kind of ensemble method usually yields performance better than any of its

constituent models (Géron, 2019). An important advantage of this method of ensemble learning is that the weights of the constituent ML algorithms can be adjusted to improve model averaging.

- Stacking: In this kind of ensemble, there is no process of voting. Instead, an ML algorithm is trained to combine the predictions of the constituent algorithms. Just like voting classifiers, this stacking ensemble also usually yields results better than its constituent models. (Wolpert, 1992)

2.2.2 Multilayer Perceptron

Multilayer Perceptron (MLP) is an artificial neural network in which the information flows through the network in one direction only. This means that the information from the first (input) layer reaches the last (output) layer without any feedback (Goodfellow et al., 2016). Due to this feature, MLPs are also called feedforward neural networks. An MLP contains layers of neurons or nodes, where each neuron propagates information to every neuron of the subsequent layer, making it a fully connected network. An MLP is made up of multiple layers of neurons, outof which the first is the input layer, and the last is the output layer. All the layers in between are called hidden layers, whose number can be varying. The ultimate goal of an MLP is to create an approximate function, that maps the input to its output. Figure 2.2 shows a pictorial representation of an MLP.

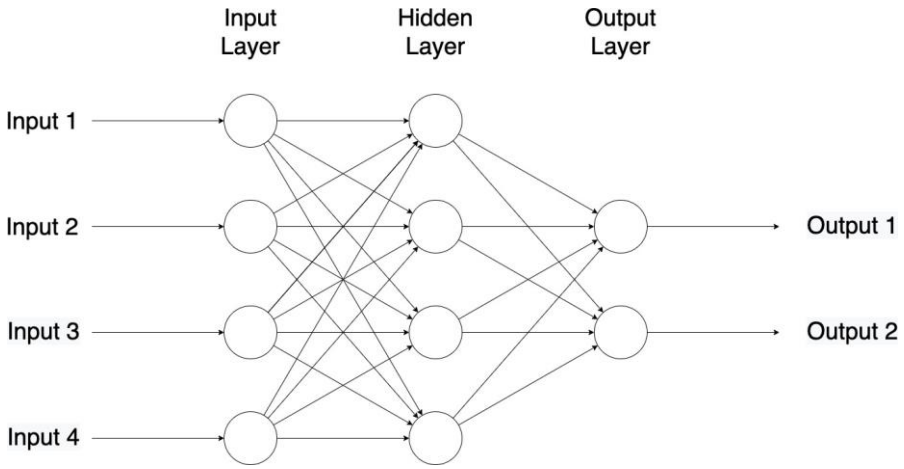

Figure 2.2. Multilayer perceptron with 1 hidden layer.

At each layer of the MLP, the input is multiplied by a weight assigned to that neuron. A bias is added, and the sum of the weighted input and the bias is passed through an activation function. Common activation functions are 1) Rectified linear unit (ReLU), 2) Sigmoid, and 3) Hyperbolic tangent function (tanh). In the context of classification tasks like intrusion detection, the input x to the MLP would be the samples of network connections and the output y would be their respective classes. (Goodfellow et al., 2016)

As with any other machine learning models, MLPs are trained by optimizing the loss function. The loss is the difference between the output predicted by the model and the actual output for a given input. The MLP approximates the mapping function f by minimizing this loss. (Goodfellow et al., 2016)

2.3 Important Inferences

Based on the literature discussed above in the previous section, following are some important points to note that shape the approach for the desired IDS model for this research:

1. According to Yin et al. (2017), deep learning has a higher potential to create intrusion detection models as compared to traditional ML models because machine learning constitutes shallow learning, which is not suitable in cases where the training data is very high-dimensional. In such cases of high-dimensional data, deep learning techniques have

better representational power. Moreover, shallow learning techniques depend on feature engineering, which is not required for training deep learning models.

2. According to a survey by Gamage et al. (2020), among the various deep learning approaches that have been used for intrusion detection, feed-forward deep neural networks achieved better results as compared to recurrent neural networks like Gated Recurrent Unitsand Long Short-Term Memory.

3. A notable observation in the previous ML based intrusion detection approaches proposed by researchers is that ensemble models of machine learning algorithms outperformed their individual counterparts by achieving higher classification accuracy and less variance.

4. Based on previous research on intrusion detection systems that used ML and deep learning, most IDS models face difficulties in accurately classifying R2L and U2R attacks. This is because in NSL-KDD dataset and other benchmark datasets for intrusion detection, the number of samples from R2L and U2R attack classes is much smaller in comparison to the number of samples from the DoS and probe attack classes. Therefore, in most cases, the performance of the intrusion detection model against R2L and U2R attacks is the critical factor in determining its efficiency.

5. Ensemble learning combines individual machine learning algorithms to achieve better prediction results than those achieved individually by the said ML algorithms. Besides this, weights on the members of the ensemble can be adjusted or learn to improve classification.

Looking at point 4, whatever ML or DL model is used for intrusion detection should strive to achieve good classification results for R2L and U2L attacks. For this reason, and also based on the above points, this research uses a Multilayer Perceptron (MLP) in an ensemble with the machine learning algorithms used by previous IDS researchers, that were effective against R2L and U2R attacks. This way, the ensemble can be trained to focus better on classifying R2L and U2R attacks. The details of this approach will be elaborated in the next chapter.

2.4 Summary

This chapter summarized the ML, ensemble, and DL models for intrusion detection given by previous IDS researchers and described their effectiveness against DoS, probe, R2L, and U2R classes of attacks. The chapter also talked about the deficiencies of each of those models and highlighted the need for a uniform intrusion detection model. Some important inferences that which help in building the required uniform model were explained.

CHAPTER 3. METHODOLOGY

This chapter talks about a methodology to explore a machine learning based approach to an IDS that can detect all classes of network attacks uniformly. It also talks about the data collection methods, performance evaluation metrics, and the proposed timeline for the experiment.

3.1 Dataset

The data required to conduct this research is obtained from a publicly available dataset called NSL-KDD (*NSL-KDD dataset*, n.d.). This dataset is a newer and refined version of the KDD Cup 1999 dataset (Tavallaee et al., 2009), which was created by the "Association for Computing Machinery" (ACM). ACM's "Special Interest Group on Knowledge Discovery and Data mining" collected flow-based connection data of over 4.9 million network connections that include normal and malicious connections. Among these malicious connections are connections from the four main attack classes – DoS, Probe, U2R and R2L. But most of these connection instances in the dataset consist of redundant records, which skewed the learning of AI models trained on this dataset towards the duplicate records (Tavallaee et al., 2009). The NSL-KDD dataset also has the same classes of connections but has fewer instances, owing to removal of redundant records. Since there are no redundant records, the dataset gives a better representation of normal and malicious connections.

The NSL-KDD dataset is meant for data mining experiments, which, along with no redundant records, makes it an ideal dataset for evaluating the proposed model. This dataset is also the benchmark dataset for research in intrusion detection as its primary purpose is to train and evaluate predictive data mining models that can classify legitimate and illegitimate connections. Each sample in the dataset has 43 features depicting the properties of the connection. These features include numerical features like connection duration, source bytes, destination bytes, etc. as well as categorical features like protocol, service, and flag. Each samplein the dataset is marked as normal or malicious – malicious attacks being labeled according to their respective types such as smurf, ipsweep, buffer-overflow, etc. There are 39 such attacks that can be categorized into four classes, namely DoS, probe, R2L, and U2R (as shown in table 3.1). Table 3.2 depicts the distribution of instances in the dataset among various attack classes. As seenfrom table 3.2, the

number of R2L and U2R instances is significantly less than that of DoS and probe attacks. Therefore, classification models need to learn the features of R2L, and mainly U2Rattacks, as they are very few in number in the dataset.

Table 3.1. *Various attacks in the NSL-KDD dataset*

DoS	Probe	R2L	U2R
processtable	ipsweep	warezmaster	rootkit
worm	portsweep	warezclient	sqlattack
udpstrom	satan	snmpgetattack	perl
pod	mscan	xsnoop	ps
mailbomb	saint	xlock	buffer_overflow
smurf	nmap	named	loadmodule
apache2		spy	xterm
neptune		sendmail	
land		multihop	
teardrop		phf	
back		snmpguess	
		guess_passwd	
		ftp_write	
		httptunnel	
		imap	

Table 3.2. *NSL-KDD dataset composition*

Instance type	Count
Normal	77,053
DoS	53,386
Probe	14,077
R2L	3,880
U2R	119
Total	148,515

3.2 Approach

This research uses a multilayer perceptron (MLP) in a soft voting ensemble (Géron, 2019) with machine learning models that effectively classify R2L and U2R attacks. These machine learning models are the ones proposed by previous researchers that were effective against either R2L and/or U2R attacks. The reasons for this approach were discussed in chapter 2. To summarize the reasons:

1. Feedforward neural networks work best in intrusion detection tasks, when compared to other kinds of neural networks like recurrent networks. Hence, the use of a multilayer perceptron.

2. An ensemble model often makes better predictions than the best individual predictor inside it. Besides, classification accuracy of the model for R2L and U2R attacks is extremely important to achieve uniform efficiency. Hence, the combination of MLP with previous ML models.

The approach used in the experiments discussed in chapter 4 is as follows: A multilayer perceptron was initially trained on the benchmark dataset for intrusion detection, NSL-KDD. Based on the results, appropriate machine learning models were combined with the MLP to form a soft-voting ensemble. For example, if the initial results of the MLP showed that it needed improvement in classifying U2R attacks, a previously researched machine learning model, which was effective against U2R, such as the random forest model given by Hasan et al. (2014), was added, and the performance of the resultant soft voting ensemble was evaluated. Now if the ensemble needed improvement in classifying another class, a model which has been proven to be effective for that class was added and so on. Soft-voting was preferred to hard-voting because unlike hard voting, where the final vote is decided by majority, soft voting operates using probabilities of the output. For every class in the dataset, each predictor predicts the probability that an instance belongs to that class. Finally, the class that receives the most cumulative vote (probability) from all predictors becomes the final output. This is better than majority voting because output probability takes vote confidence into consideration and hence, the votes with low confidence (or probabilities) receive low importance and vice versa (Géron, 2019). The ensemble

was further tuned to focus on tricky classes like R2L and U2R by adding more weight to their instances. This entire process is discussed in detail in Chapter 4.

3.3 Experiment Structure

This section talks about how the experiment discussed in the next chapter (chapter 4) is outlined. The experiment is iterative in nature as it is performed repetitively to improve results. Below is an ordered list of the various stages of the proposed experiment:

1. Data pre-processing
2. Training the MLP
3. Testing the MLP
4. Creating the soft-voting ensemble
5. Adding to soft-voting ensemble to improve results

The following subsections explain in detail, what each stage of the experiment entails.

3.3.1 Data pre-processing

As mentioned in section 3.1, the dataset used for this experiment is NSL-KDD dataset (*NSL-KDD dataset*, n.d.). It is publicly available and can be downloaded from the internet. The first step of the experiment was data pre-processing. Pre-processing involved procedures like feature scaling and converting categorical variables into numerical variables. Many machine learning algorithms work with numerical features. So, it is important to convert categorical features into numerical features so that there is no loss of information for the machine learning model. This conversion can be done by one-hot encoding, which expresses the categorical features as binary values (Géron, 2019). The details of this procedure will be discussed in chapter 4. After pre-processing the data, the final step was to split the dataset into training and testing sets (more details in chapter 4).

3.3.2 Training the MLP

After splitting the pre-processed NSL-KDD dataset into training and testing sets, the first step was to train the Multilayer Perceptron on the training set. During training, the model was validated using K-fold cross validation, which is a re-sampling technique to assess machine learning models, where the training data is split randomly into k equal parts, and the machine learning model is trained on K-1 parts of the training set, and its learning is validated using the one remaining part. This is repeated k times for all k combinations of training and validation sets. Chapter 4 elaborates on the training procedure and MLP configuration.

3.3.3 Testing the MLP

After training and validating the MLP, its performance was evaluated on the test set using the analysis methods discussed in section 3.4. If there was any class of attacks that needed improvement in classification accuracy, an appropriate machine learning model was added to create a soft-voting ensemble, as discussed in the approach in section 3.2.

3.3.4 Creating the soft-voting ensemble

This step of the experiment involved adding a machine learning model to the MLP to improve its performance against the most challenging attack class. After assessing the test results of the MLP, based on what class of attack needs better classification accuracy, an appropriate machine learning model, which is proven to work by previous researchers, was chosen for addition to the MLP to create a soft-voting ensemble. For example, if the model under-performedagainst U2R attacks, a random forest was added to the MLP as it was previously proven to work against U2R attacks by Hasan et al. (2014), and the resulting ensemble was again trained and tested.

3.3.5 Adding to soft-voting ensemble to improve results

The soft-voting ensemble created in the previous step of the experiment was trained on the training set, validated using K-fold cross validation, and its performance was evaluated on the test set. Based on the results, if a class of attacks still needed improvement in classification accuracy, another machine learning model, just like in the previous step, was added to the ensemble.

Chapter 4 talks about this procedure in more detail.

Figure 3.1 shows a flowchart depicting the stages of the experiment.

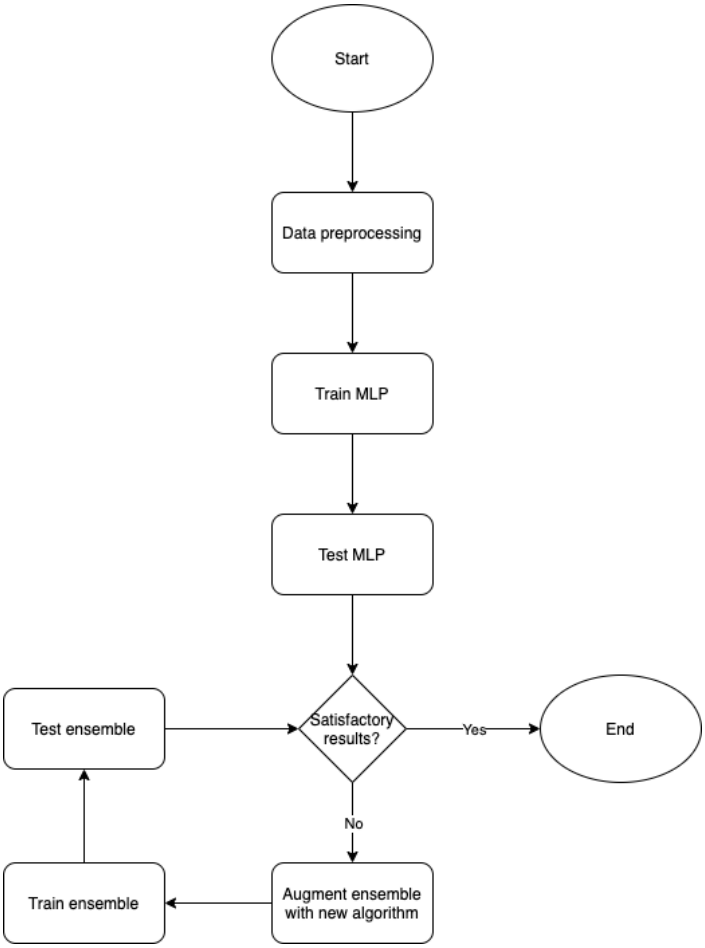

Figure 3.1. Flowchart depicting the steps involved in the experiment

3.4 Analysis Methods

As the purpose of this research is to develop an IDS model that can classify all kinds of network intrusions with uniform efficiency, the most important performance metric in this research is the classification efficiency of the proposed IDS model for *each class* of network intrusion. This is defined as the percentage of attack instances correctly classified for each class of attack. This metric is also called as *recall*. Another evaluation metric that can judge the model's performance is the overall *precision* of the ML model. Precision is defined as the percentage of correct classifications done by the model. To visually depict precision and recall ofthe ML model, a *confusion matrix* is used. Another important factor to consider while evaluatingthe machine learning model is the count of false positives. Since the task of an IDS is to identifyattacks in network traffic, false alarms can cause unnecessary problems. Therefore, a good intrusion detection model should have few false positives.

To determine if the model developed in this research was successful, the minimum required recall for each class of attack was set to 80%. This way, there is a guarantee of a minimum classification efficiency against every class of attacks, thus making the machine learning model *uniformly* effective against all attack classes.

CHAPTER 4. EXPERIMENT AND RESULTS

This chapter talks about the experiments conducted in this research and the obtained results. The chapter also discusses the various combinations of soft-voting ensemble machinelearning models and their respective results.

4.1 Experiment

As discussed in chapter 3, to build a uniform intrusion detection model, a multlilayer perceptron was trained and evaluated on the NSL-KDD dataset initially. Based on the evaluationresults, if the MLP needed to perform better against a certain attack class, the ML algorithms thatwere proven by previous researchers to be efficient against that attack class, were trained and tested in an ensemble with the MLP. Different ML algorithms were added or removed from the ensemble to improve results.

The experiments involved in the research were conducted with the help of Weka (Holmes,Donkin, & Witten, 1994), which is an open-source software for ML and data mining. Weka wasused for data-preprocessing as well as building the machine learning models. As discussed in chapter 3, the experiment started with pre-processing the NSL-KDD dataset. The categorical features in the dataset were converted to numerical by one-hot encoding, which is a method of converting the categorical feature values into binary values. For example, the 'protocol' feature inthe dataset takes three categorical values, namely tcp, udp, and, icmp. After one-hot encoding, this 'protocol' feature was replaced by three features – tcp, udp, and icmp. The values of these features are 1's and 0's, with 1 meaning that the feature is present in the instance, while 0 meaning that the feature is absent. This is shown in tables 4.1 and 4.2. Table 4.1 shows a section of the NSL-KDD dataset before one-hot encoding. The instances have protocol values as tcp, icmp, or udp. Table 4.2 shows the same section of the dataset after one-hot encoding. Notice howeach value under the protocol feature became a feature of its own with binary values.

The numerical attributes were normalized for feature scaling. After data pre-processing, the dataset was divided into two parts – training and testing sets. 80% of the total instances in thedataset formed the training set while the remaining 20% formed the testing set. These

training and testing sets files were converted to .arff format for better compatibility with Weka. The distribution of instances of various attack classes in the training and test sets is shown in tables 4.3and 4.4.

Table 4.1. *A section of the dataset before one-hot encoding of protocol variable.*

src bytes	protocol
300	tcp
520	icmp
54	udp
76944	tcp

Table 4.2. *Section of the dataset shown in table 4.1 after one-hot encoding of protocolvariable.*

src bytes	tcp	udp	icmp
300	1	0	0
520	0	0	1
54	0	1	0
76944	1	0	0

Table 4.3. *Training set composition*

Instance type	Count
Normal	61,509
DoS	42,731
Probe	11,351
R2L	3,125
U2R	96
Total	118,812

After splitting the NSL-KDD dataset into training and testing sets, the multilayer perceptron was trained on the training set. The MLP was designed based on the feature variables and labels in the dataset. Each of the 42 features or attributes in the dataset was an input to the MLP. Hence, the number of neurons in the input layer of the MLP was 42. The number of neuronsin the output layer of the MLP was decided by the number of classes or labels in the dataset.

Since the dataset has 5 classes – normal, DoS, probe, U2R, and R2L, the number of neurons in theoutput layer was 5. Weka sets the number of hidden layers in the MLP to the average of number of features (42) and number of classes in the training data (5) by default. Therefore, the MLP in this experiment had 23 hidden layers. The MLP was trained over the training set and validated using k-fold cross validation with 6 folds for 50 epochs. The trained MLP was then tested on thetest set and the results were observed using a confusion matrix which is shown in table 4.5.

Table 4.4. *Testing set composition*

Instance type	Count
Normal	15,544
DoS	10,655
Probe	2,726
R2L	755
U2R	23
Total	29,703

Table 4.5. *Confusion matrix of the MLP*

	Normal	DoS	R2L	Probe	U2R
Normal	15259	14	63	205	3
DoS	2	10607	2	44	0
R2L	5	2	721	25	2
Probe	9	2	3	2712	0
U2R	0	0	4	10	9

Based on the confusion matrix above, the MLP performed well in classifying instances ofnormal connections, DoS, probe, and R2L attacks but underperformed against U2R attacks. The recall values for each class is given in table 4.6.

Table 4.6. MLP Recall

Instance type	Recall
Normal	98.2%
DoS	99.5%
Probe	99.5%
R2L	95.5%
U2R	39.1%

Every class except U2R had a recall value of over 80% which was the minimum requiredvalue to achieve the desired model as discussed in chapter 3. Since the model required improvement in performance against U2R class of attacks, it needed to be augmented with machine learning models used by previous researchers, that were effective against U2R attacks.From table 2.1, it can be seen that SVM, Random Forest, and Decision Tree were the algorithmseffective against U2R attacks. Therefore, the MLP would be trained in a soft-voting ensemble with each of these three algorithms. The results of the different combinations of the ensembles will be discussed in the following subsections.

4.1.1 MLP + SVM

The first machine learning algorithm trained in a soft-voting ensemble with the MLP was SVM. The SVM was built with a radial basis function kernel with the help of Weka. Only a thirdof the original dataset was used to reduce the time taken for training. Based on the how the modelperforms on the partial dataset, the best model would be chosen and evaluated on the full dataset.This partial dataset was divided into training and testing sets using an 80-20 split. The ensemble was trained on the training data after synthetically up-sampling the minority class, i.e., U2R, using “Synthetic Minority Oversampling Technique” (SMOTE) (Chawla et al., 2002). The reasonbehind this was that most machine learning algorithms under-perform against the minority class due to learning from only few instances. To better learn the features of the minority class instances, SMOTE was used. Although Generative Adversarial Networks (GANs) (Goodfellow etal., 2014) could also be used to synthesize minority class instances, the computational cost to build and train a GAN is very high (Creswell et al., 2018). For this reason, SMOTE was preferred as the upsampling technique. The training time was high as the SVM was slow in computation. The performance of this ensemble model on the test set is depicted in the confusionmatrix in table 4.7.

Table 4.7. *Confusion matrix of MLP + SVM*

	Normal	DoS	R2L	Probe	U2R
Normal	4601	3	2	8	0
DoS	136	3227	2	8	0
R2L	53	0	550	5	0
Probe	100	39	0	799	0
U2R	14	0	0	0	0

As seen, the model's performance against U2R instances worsened as no instance of U2Rclass was correctly classified. Hence, this model, along with the current SVM, needed to be augmented with another machine learning algorithm, i.e., either random forest or decision tree. Inthe next experiment (section 4.1.2), the MLP + SVM soft-voting ensemble was augmented with aRandom Forest.

4.1.2 MLP + SVM + Random Forest

A random forest with 100 trees was added to the soft-voting ensemble using Weka. Thenew ensemble of MLP, SVM, and Random Forest yielded better results against U2R instanceswith a significant increase in recall. The confusion matrix in table 4.8 depicts how the model performed on the same test set used in the previous subsection.

Table 4.8. *Confusion matrix of MLP + SVM + Random Forest*

	Normal	DoS	R2L	Probe	U2R
Normal	4606	1	4	3	0
DoS	4	3366	0	3	0
R2L	8	0	596	0	4
Probe	12	1	1	924	0
U2R	6	0	1	0	7

7 out of 14 instances of U2R were correctly classified, meaning the recall was 50%. Eventhough the recall increased with the addition of Random Forest, it was still below 80% and the model was extremely slow in training. For this reason, this chain of ensemble model was not extended further and SVM was not considered for subsequent models.

4.1.3 MLP + Random Forest

The next combination of soft-voting ensemble that was tried was MLP with a random forest. Like in the previous experiment, the minority class (U2R) was upsampled before training.The ensemble model was trained, validated using k-fold cross validation, and evaluated on the testset. From the confusion matrix in table 4.9, the recall for this model is 10 out of 14, i.e., 71%, which is higher than that of the previous models. The training speed also increased with the exclusion of SVM. However, the recall can still be improved. For this reason, another pre-processing technique called class-balancing (Laurikkala, 2001) was tried. This is a method ofadding weights to instances in the dataset so that each class has the same total weight. This way, the minority class instances have higher weights as compared to the majority class instances. The performance of the ensemble with class-balancing is shown in the confusion matrix in table 4.10:

Table 4.9. *Confusion matrix of MLP + Random Forest with SMOTE*

	Normal	DoS	R2L	Probe	U2R
Normal	4570	8	25	10	1
DoS	1	3369	0	2	1
R2L	5	0	596	0	7
Probe	6	8	5	916	3
U2R	0	0	4	0	10

Table 4.10. *Confusion matrix of MLP + Random Forest with class balancing*

	Normal	DoS	R2L	Probe	U2R
Normal	3753	0	0	0	861
DoS	0	2868	0	0	505
R2L	0	0	464	0	144
Probe	0	0	0	736	202
U2R	0	0	0	0	14

Class balancing helped in achieving a perfect 100% recall for U2R instances but caused asignificant number of instances from other classes to be classified as U2R, thus drastically diminishing the model's precision. Because of the high amount of false positives for U2R attacks,this model is deemed to be poor although it achieves 100% recall.

To mitigate false positives, class balancing and SMOTE were used together. This approach could mitigate false positives because balancing the classes after upsampling the minority class results in a lower additional weight on the minority class instances, resulting in lower bias towards them. The test results of the MLP + random forest ensemble on this new processed training data are shown in table 4.11.

Table 4.11. *Confusion matrix of MLP + Random Forest with SMOTE + classbalancing*

	Normal	DoS	R2L	Probe	U2R
Normal	4433	36	81	63	1
DoS	4	3341	12	15	1
R2L	1	0	571	3	33
Probe	1	6	17	912	2
U2R	0	0	3	0	11

The false positives have significantly reduced and the recall is better than the one achievedwith just upsampling but it is still below 80%.

4.1.4 MLP + Decision Tree

After training the MLP in combinations of soft-voting ensembles involving SVM and random forest, the final remaining algorithm, decision tree was tried. This was a J48 decision tree(Quinlan, 2014). Without SMOTE, on the full dataset, the MLP + J48 ensemble learning model gave the following results (see table 4.12).

Table 4.12. . *Confusion matrix of MLP + J48*

	Normal	DoS	R2L	Probe	U2R
Normal	15503	8	12	19	2
DoS	4	10643	1	6	1
R2L	12	1	732	3	7
Probe	6	2	1	2717	0
U2R	0	0	4	1	18

The recall for U2R instances was 78% without any up-sampling used while training. Thetraining speed of the model was also higher than all the previous ensembles, making it a good fitfor the desired model. With upsampling the minority class, the recall increased to 87% as seen inthe confusion matrix in table 4.13.

This soft-voting ensemble model resulted in a recall that was well above the required 80%. The overall precision was 99.6% and overall false positive percentage was 0.1%. Since the performance objectives discussed in section 3.4 of chapter 3 were met, this model was fit to be thefinal required model in this research.

Table 4.13. *Confusion matrix of MLP + J48 with SMOTE*

	Normal	DoS	R2L	Probe	U2R
Normal	15489	14	17	21	3
DoS	21	10642	2	7	1
R2L	9	1	736	6	3
Probe	8	1	0	2714	3
U2R	0	0	3	0	20

Table 4.14 shows a summary table showing the recall values for U2R as well as all otherclass instances achieved by all the models tried in this experiment.

Table 4.14. *Summary of the various models evaluated in this research*

Model	Recall (in %)				
	Normal	DoS	Probe	R2L	**U2R**
MLP	98.2	99.5	99.5	95.5	**39.1**
MLP + SVM	99.7	95.6	85.1	90.4	**0**
MLP + SVM + Random Forest	99.8	99.8	98.5	98.0	**50.0**
MLP + Random Forest	96.1	99.1	97.6	93.9	**78.6**
MLP + J48 (with SMOTE)	99.6	99.7	99.6	97.5	**86.9**

The MLP + J48 soft-voting ensemble model was built again using Python (Keras + Scikit-learn) and evaluated to see if there was any improvement in performance. While the confusion matrix was similar, the training speed was significantly higher as compared to Weka's(as shown in table 4.15).

Table 4.15. *Weka vs Python*

Tool	Training time (in seconds)
Weka	782.35
Python	611.94

The time taken by the model to classify a new unknown instance was calculated usingPython's time library. The time() method in Python's time library retrieves the current time.

Using this method, the current time was recorded just before testing the ensemble model on the testing set and once again just after finishing testing. The difference between the two time valueswill give the training time. This time depends on the underlying hardware. This experiment used a 2.7 GHz Dual-Core Intel Core i5 processor and the average classification time for the model was 23.75 microseconds. This testing time was calculated 10 times and averaged out. The meanclassification time per new instance is given in table 4.16.

Table 4.16. *Statistics of classification time for new instance*

Mean	23.75 μs per instance
Standard deviation	1.79 μs per instance

CHAPTER 5. DISCUSSION AND CONCLUSION

This chapter talks about some concluding remarks on this research and discusses if the objectives of the research were met and research questions were answered. It also talks about the feasibility of the intrusion detection model in the real world and throws light on scope for further research.

5.1 Discussion

Based on the experiments conducted in this research, a soft-voting ensemble learning model consisting of a multilayer perceptron and a J48 decision tree emerged as an ML model for uniform intrusion detection. The model trained using Weka achieved more than 80% detection rate for all four of DoS, Probe, R2L, and U2R classes of attacks, overcoming the classification barriers faced by machine learning based intrusion detection models given by previous IDS researchers. The time taken for training the model using Weka was about 13 minutes which couldbe reduced to 10 with the help of Python and its Keras and Scikit-learn libraries. This training time can be further reduced using a hardware accelerator like the GPU (Graphics Processing Unit). The presence of a decision tree in the MLP + J48 ensemble model tackles the problem of class imbalance in the dataset used in this research. This is because decision trees have a hierarchical structure due to which they are even able to learn the characteristics of the minority class with good efficiency (*How to Handle Imbalanced Classes in Machine Learning*, 2020). In addition, upsampling techniques like SMOTE help in synthesizing minority class instances so thatthe model better learns the features of the minority class.

The significance of this research is that network security administrators need not worry about the kind of network intrusion detection model to use when they do not know what attacks to anticipate. Instead, they have one model that is effective against all four classes of network attacks, and they can work on simply improving its performance in the future.

5.1.1 Answers to research questions

In chapter 1 of this document, there were two research questions that were posed. The questions addressed two important problems involved in achieving uniform efficiency across various attack classes by intrusion detection models.

Research question 1: Is is possible to have an artificial intelligence based IDS model thatcan classify all the four kinds of network intrusions with uniform efficiency?

Answer: Yes. Based on the experiments conducted in the research and the final MLP + J48 ensemble learning model which achieved detection rates of at least 87% for each of DoS, probe, R2L, and U2R attack classes, it can be seen that it is possible to have an AI based IDS model thatis uniformly effective against all classes of network attacks.

Research question 2: Would there be a trade-off between achieving uniform efficiencyacross all attack classes and achieving high efficiency for individual classes?

Answer: Based on the evaluations of the various models tried out in the experiments, this trade-off came into play only when the training data was class-balanced. The model suffered fromtoo many false positives as many instances from other classes were being classified as minority class instances. As seen in table 4.8, the confusion matrix shows that 861 normal connection instances, which are about 19% of the total normal instances, were classified as attacks. This problem was eliminated by removing class balancing and changing the algorithms used in the ensemble from random forest to J48 decision tree.

Therefore, the answer to this question is: While there can be trade-off between achieving uniform efficiency across all attack classes and achieving high efficiency for individual classes, the trade-off can be avoided using the right ML algorithms in the intrusion detection model. As seen in chapter 4 section 4.1.4, the final MLP + J48 model did not have this trade-off and could successfully achieve high recall rates for all attack classes.

5.1.2 Practical Consideration

As discussed in chapter 4, the average time taken by the MLP + J48 soft-voting ensemble model to detect a new unknown connection instance was 23.75 microseconds. To check if this speed is practical and the model can be used in real world intrusion detection systems, its detection speed was compared to the speeds of two popular IDS software used in the real world –Snort (Roesch et al., 1999) and Suricata (*Suricata*, n.d.). Snort is a popular and widely used open source rule-based IDS, which is now developed by Cisco. Suricata is also an open source IDS but is developed by "Open Information Security Foundation". According to Shah and Issac (2018), Snort's processing power is 31,333 packets per second, which means that it takes 31.92 microseconds to process a single packet, and Suricata's processing power is 52,333 packets per second, which is 19.11 microseconds per packet. The MLP + J48 ensemble learning model built in this research could detect a new connection instance in 23.75 microseconds, which is comparable to the detection speeds of Snort and Suricata. Therefore, this model might be useful in a real world intrusion detection system.

Table 5.1. *Detection time comparison*

Model/Software	Detection time per packet/connection (in μs)
Snort	31.92 per packet
Suricata	19.11 per packet
MLP + J48	23.75 per connection

5.2 Conclusion

Artificial intelligence based intrusion detection systems developed by previous IDS researchers, although effective overall, were not uniformly effective against different kinds of attacks. The need for an artificial intelligence based uniform intrusion detection model is important as it allows network security admins to use a 'one-for-all' intrusion detection model toprotect their computer networks from all kinds of malicious connections, instead of having to choose the right model. By leveraging the power of deep neural networks and incorporating themin ensemble learning models, based on a simple idea that ensemble learning models yield more

efficient models than their individual counterparts, this uniform intrusion detection model was made possible. The developed model was carefully evaluated and found to achieve good detection rates for all the attack classes considered in this research. Besides, the model resulted in very few false positives and high overall precision. This uniform intrusion detection model provides opportunities for further research into how to further increase its efficiency as it is incorporated in the real world.

REFERENCES

Aburomman, A. A., & Reaz, M. B. I. (2016). A novel svm-knn-pso ensemble method for intrusion detection system. Applied Soft Computing, 38, 360–372.

Amor, N. B., Benferhat, S., & Elouedi, Z. (2004). Naive bayes vs decision trees in intrusion detection systems. In Proceedings of the 2004 acm symposium on applied computing (pp. 420–424).

Barlow, H. B. (1989). Unsupervised learning. Neural computation, 1(3), 295–311.

Bebis, G., & Georgiopoulos, M. (1994). Feed-forward neural networks. IEEE Potentials, 13(4), 27–31.

Bishop, C. M. (2006). Pattern recognition and machine learning. springer.
Cole, E. (2011). Network security bible (Vol. 768). John Wiley & Sons.

Cortes, C., & Vapnik, V. (1995). Support vector machine. Machine learning, 20(3), 273–297.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. IEEE Signal Processing Magazine, 35(1), 53–65.

Dietterich, T. G., et al. (2002). Ensemble learning. The handbook of brain theory and neural networks, 2, 110–125.

Forouzan, B. A. (2007). Cryptography & network security. McGraw-Hill, Inc.

Friedman, J. H. (1991). Multivariate adaptive regression splines. The annals of statistics, 1–67.

Gamage, S., & Samarabandu, J. (2020). Deep learning methods in network intrusion detection: A survey and an objective comparison. Journal of Network and Computer Applications, 169, 102767.

Géron, A. (2019). Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1) (No. 2).

MIT press Cambridge.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

Hasan, M. A. M., Nasser, M., Pal, B., & Ahmad, S. (2014). Support vector machine and random forest modeling for intrusion detection system (ids). Journal of Intelligent Learning Systems and Applications, 2014.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527–1554.

Holmes, G., Donkin, A., & Witten, I. H. (1994). Weka: A machine learning workbench. In Proceedings of anziis'94-australian new zealnd intelligent information systems conference (pp. 357–361).

How to handle imbalanced classes in machine learning. (2020, May). Retrieved from https://elitedatascience.com/imbalanced-classes

Keller, J. M., Gray, M. R., & Givens, J. A. (1985). A fuzzy k-nearest neighbor algorithm. IEEE transactions on systems, man, and cybernetics(4), 580–585.

Kim, D. S., & Park, J. S. (2003). Network-based intrusion detection with support vector machines. In International conference on information networking (pp. 747–756).

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. In Conference on artificial intelligence in medicine in europe (pp. 63–66).

Lee, J.-H., Lee, J.-H., Sohn, S.-G., Ryu, J.-H., & Chung, T.-M. (2008). Effective value of decision tree with kdd 99 intrusion detection datasets for intrusion detection system. In 2008 10th international conference on advanced communication technology (Vol. 2, pp.1170–1175).

Lin, W.-C., Ke, S.-W., & Tsai, C.-F. (2015). Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-based systems, 78, 13–21.

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. Applied Sciences, 9(20), 4396.

Mitchell, M. (1998). An introduction to genetic algorithms. MIT press.

Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. IEEE network, 8(3), 26–41.

Nsl-kdd dataset. (n.d.). Retrieved from https://www.unb.ca/cic/datasets/nsl.html

Quinlan, J. R. (2014). C4. 5: programs for machine learning. Elsevier.

Roesch, M., et al. (1999). Snort: Lightweight intrusion detection for networks. In Lisa (Vol. 99, pp. 229–238).

Rowland, C. H. (2002, June 11). Intrusion detection system. Google Patents. (US Patent 6,405,318)

Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to snort system. Future Generation Computer Systems, 80, 157–170.

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

Stein, G., Chen, B., Wu, A. S., & Hua, K. A. (2005). Decision tree classifier for network intrusion detection with ga-based feature selection. In Proceedings of the 43rd annual southeast regional conference-volume 2 (pp. 136–141).

Suricata. (n.d.). Retrieved from https://suricata.io/

"why do we need network security?". (n.d.). Retrieved from https://www.ecpi.edu/blog/why-do-we-need-network-security

Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241–259.

Yang, Y., Zheng, K., Wu, C., Niu, X., & Yang, Y. (2019). Building an effective intrusion detection system using the modified density peak clustering algorithm and deep belief networks. Applied Sciences, 9(2), 238.

Zadeh, L. A. (1996). Fuzzy logic, neural networks, and soft computing. In Fuzzy sets, fuzzy logic, and fuzzy systems: Selected papers by lotfi a zadeh (pp. 775–782). World Scientific.

APPENDIX A. MODEL CODE

Below is the code used to build, train, and test the MLP + J48 ensemble learning model in Python.

```
import pandas as
pd import numpy
as np import
tensorflow as tf
import time
import statistics
from keras.models import
Sequentialfrom keras.layers
import Dense
from sklearn.metrics import
confusion_matrix from
sklearn.ensemble import
VotingClassifier from sklearn.tree
import DecisionTreeClassifier


# Reading the training data into a dataframe and dividing it into
input matrix# X and label matrix y.
```

```
dataframe_train =
pd.read_csv('/content/Full_Train.xls') array =
dataframe_train.values
x1 =
array[:,0:38]
x2 =
array[:,39:40] y
= array[:,38:39]
X = np.concatenate((x1, x2), axis=1)
X_tensor = np.asarray(X).astype('float32') #Converting X from an array
to a
                                        #tensor as the MLP built
                                        using #Keras takes a
                                        tensor as input.

# Reading the testing data into a dataframe and dividing it into
input matrix# X_test and label matrix y_test.

dataframe_test =
pd.read_csv('/content/Full_Test.xls') array1 =
dataframe_test.values
x1_test = array1[:,0:38]
x2_test =
```

```
array1[:,39:40]y_test =
array1[:,38:39]
X_test = np.concatenate((x1_test, x2_test),
axis=1)            X_tensor_test          =
np.asarray(X_test).astype('float32')


# Function to build the MLP with 23
layers. def build_nn():
  mlp = Sequential()
  mlp.add(Dense(39,            kernel_initializer='he_uniform',
  activation='relu'))                         mlp.add(Dense(39,
  kernel_initializer='he_uniform',              activation='relu'))
  mlp.add(Dense(39,            kernel_initializer='he_uniform',
  activation='relu'))                         mlp.add(Dense(39,
  kernel_initializer='he_uniform',              activation='relu'))
  mlp.add(Dense(39,            kernel_initializer='he_uniform',
  activation='relu'))                         mlp.add(Dense(39,
  kernel_initializer='he_uniform',              activation='relu'))
  mlp.add(Dense(39,            kernel_initializer='he_uniform',
  activation='relu'))                         mlp.add(Dense(39,
  kernel_initializer='he_uniform',              activation='relu'))
  mlp.add(Dense(39,            kernel_initializer='he_uniform',
  activation='relu'))                         mlp.add(Dense(39,
  kernel_initializer='he_uniform',              activation='relu'))
```

```
mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu')) mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))
mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu')) mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))
mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu')) mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))
mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu')) mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))
mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))

mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu')) mlp.add(Dense(39, kernel_initializer='he_uniform', activation='relu'))
mlp.add(Dense(5, activation='sigmoid'))' mlp.compile(loss='categorical_crossentropy', optimizer='adam')
return mlp
```

```
# Scikit-learn wrapper for the MLP so that it can be used
with the# voting classifier built using sklearn.
model1 = tf.keras.wrappers.scikit_learn.KerasClassifier(build_nn,
                                                        epochs=50,
                                                        verbose=False)
model1._estimator_type = "classifier"

model2 = DecisionTreeClassifier() # Creating the J48

decision tree.classifiers = []
classifiers.append((’mlp’,
model1))
classifiers.append((’j48’,
model2))


# Creating the voting ensemble.
voting =
        VotingClassifier(estimators
        =classifiers, voting=’soft’,
        flatten_transform=True)


# Training the voting
ensemble startTime =
time.time()
```

```
voting.fit(X_tensor,      y)
stopTime = time.time()
print(f"Training time: {stopTime - startTime}s")

# Testing the voting ensemble and printing the confusion matrix

start = time.time()
y_pred_voting                              =
voting.predict(X_tensor_test)   stop     =
time.time()
confusion_matrix(y_test, y_pred_voting, labels=["normal", "dos",
"probe",
                                                  "r2l", "u2r"])
print(f"Testing time: {stop - start}s")

# Calculating the classification time per
instance.testing_times = []
detection_tim
es = []for i in
range(10):
  start = time.time()
```

```
    y_pred_voting                    =
    voting.predict(X_tensor_test)  stop    =
    time.time()
    testing_time = stop - start
    detection_time  =  testing_time  /  29703  *
    1000                *                1000
    testing_times.append(testing_time)
    detection_times.append(detection_time)


# Printing the mean and standard deviation of classification times.
avg_detection_time = statistics.mean(detection_times)
std_dev  =  statistics.stdev(detection_times)
print(f"Average  detection  time  =  {avg_detection_time}
microseconds") print(f"Standard  deviation  =  {std_dev}
microseconds")
```