Comparative Analysis of Machine Learning Techniques for IoT Anomaly Detection Using the NSL-KDD Dataset

Zaryn Good†, Waleed Farag†, Xin-Wen Wu††, Soundararajan Ezekiel†, Maria Balega†, Franklin May†, and Alicia Deak†

knbbc@iup.edu farag@iup.edu xwu@umw.edu sezekiel@iup.edu cclz@iup.edu wyxy@iup.edu vsscc@iup.edu

†Department of Mathematical and Computer Sciences, Indiana University of PA, Indiana, PA, USA

††Department of Computer Science, University of Mary Washington, Fredericksburg, VA, USA

Summary

With billions of IoT (Internet of Things) devices populating various emerging applications across the world, detecting anomalies on these devices has become incredibly important. Advanced Intrusion Detection Systems (IDS) are trained to detect abnormal network traffic, and Machine Learning (ML) algorithms are used to create detection models. In this paper, the NSL-KDD dataset was adopted to comparatively study the performance and efficiency of IoT anomaly detection models. The dataset was developed for various research purposes and is especially useful for anomaly detection. This data was used with typical machine learning algorithms including eXtreme Gradient Boosting (XGBoost), Support Vector Machines (SVM), and Deep Convolutional Neural Networks (DCNN) to identify and classify any anomalies present within the IoT applications. Our research results show that the XGBoost algorithm outperformed both the SVM and DCNN algorithms achieving the highest accuracy. In our research, each algorithm was assessed based on accuracy, precision, recall, and F1 score. Furthermore, we obtained interesting results on the execution time taken for each algorithm when running the anomaly detection. Precisely, the XGBoost algorithm was 425.53% faster when compared to the SVM algorithm and 2,075.49% faster than the DCNN algorithm. According to our experimental testing, XGBoost is the most accurate and efficient method.

Keywords:

Internet of Things (IoT), XGBoost, SVM, DCNN, NSL-KDD, Machine Learning, Anomaly Detection

1. Introduction

Internet of Things (IoT) is a blend of two words “Internet” and “Things”. The Internet is a computer network that provides a variety of communication facilities and information globally. The interconnected computer networks use the standard Internet protocol suite (TCP/IP). On both a global and local level it consists of millions of private, public, business, and government networks to create a whole network of networks. With billions of users every day, the Internet has become staple in cultures and society across the world. While majority of the population has at least some concept of what the Internet is, when you bring up the topic of “Things”, that could really mean anything. The “Things” are tangible objects that are present in this physical or material world. At its core, the Internet of Things is interconnection through the internet of objects embedded with a computing device allowing them to both receive and send data [1]. Various IoT systems are being increasingly applied to a variety of domains, such as, Industry 4.0, smart energy, and smart cities and buildings to name a few. While the IoT with its expanding applications offers many opportunities to our society, they also bring highly challenging cybersecurity issues. The IoT devices connected to the Internet without any built-in security mechanisms create easy-to-use gateways and a larger attack surface for attackers and pose a great cybersecurity risk [1, 2].

The goal of this study is to build effective and efficient anomaly detection models which can work in dynamic environments. As networking environments may be dynamic and network traffic captures may be different, it is important to comparatively study the detection models in regard to different datasets. In our previous work [2], we studied anomaly detection models built on XGBoost, SVM, and DCNN working with the IoT-23 dataset. These models were chosen as they are the most typical supervised machine learning techniques used for classification purposes. Using these machine learning models, anomalies can be classified which can help assess IoT devices to improve their security. In this paper, our research was focused on these models in regard to the NSL-KDD dataset.

The NSL-KDD dataset was chosen as it was designed to be applied as an effective benchmark dataset for the use of researchers to compare different intrusion detection methods. The dataset is widespread and has been used to train many models, allowing for effective comparison and benchmarking against previously trained models’ results. Further, the size of the dataset allows for faster train times and is more affordable [3].

The NSL-KDD dataset is an improved version of the KDD CUP ’99 dataset which was built based on the data captured in DARPA’98 IDS evaluation program [4]. DARPA’98 is comprised of about 4 gigabytes of compressed binary tcpdump data of 7 weeks of network traffic. The KDD CUP ’99 training data consists of around 4,900,000 entries containing 41 features and is labeled as

Manuscript received January 5, 2023
Manuscript revised January 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.1.7

either normal or an attack [3].

Although the KDD CUP ’99 dataset has been used for many years, it consists of several inherent problems. For the background and attack data they used synthesized data when creating the dataset. The workload of this data does not seem similar to traffic that were to occur in real networks. The dataset also contains no exact definition for the attacks. They are only categorized as normal or attack rather than the name of the specific attack [5]. The KDD CUP ’99 dataset also has a large number of redundant records as about 78% and 75% are duplicated in the train and test sets respectively [6]. To amend these issues, the NSL-KDD dataset was created.

This dataset differs from the IoT-23 dataset in that it has less data overall, however it contains more classifications. Both datasets have imbalanced data which is an issue. To cope with this issue, the data was assessed using balanced accuracy as this evaluation metric takes this into account [3].

In our research, each algorithm was assessed based on accuracy, precision, recall, and F1 score. Our research results show that the XGBoost algorithm outperformed both the SVM and DCNN algorithms achieving the highest accuracy. We further studied the efficiency of the anomaly detection models powered by these ML algorithms and obtained interesting results on the execution time taken for each model when running the anomaly detection. Our research showed that the XGBoost algorithm was 425.53% faster when compared to the SVM algorithm and 2,075.49% faster than the DCNN algorithm. According to our experimental testing, XGBoost is the most accurate and efficient method.

Comparatively showing the effectiveness and efficiency of ML powered anomaly detection models working on a different dataset, this study will help individual users and organizations identify the most effective and efficient anomaly detection systems powered by machine learning algorithms in securing their IoT systems and devices based on the working environments and types of data.

The rest of the paper is organized as follows. In section 2, related works to this study will be presented. In section 3, machine learning techniques used in this study will be explained. In section 4, the experimental consideration will be addressed to include preprocessing of data and training of machine learning models. In section 5, evaluation metrics will be addressed, and the results from each of the machine learning models will be compared. The concluding remarks will be presented in section 6.

2. Related Works

Recent work has shown the effectiveness of machine learning and its ability to help with the security of IoT devices. In 2016, Canedo and Skjellum conducted research involving IoT devices using Artificial Neural Networks supporting the effectiveness of using machine learning to secure IoT devices [7]. Further, it was determined that with the great amount of data being collected from IoT devices, new algorithms are needed for extracting data and applying regression and classification for improving security [8].

In one of our recent works, we applied XGBoost, SVM, and DCNN to the IoT-23 dataset to study how well these machine learning algorithms could classify anomalies. The data was preprocessed having 4 total classifications where 50% of the data was malicious and 50% of the data was benign. After training each of the machine learning models with this data, XGBoost proved to be most effective and efficient method when working on the IoT-23 dataset [2].

However, it is interesting to ask: What are the optimal anomaly detection models when working in different environments with different datasets? Due to diverse IoT applications, networking environments may be dynamic and network traffic may be different. Therefore, it is important to comparatively study the detection models regarding different datasets. Extending our previous research on the IoT-23 dataset, in this paper, we investigate the anomaly detection models in regard to the NSL-KDD dataset, attempting to identify the most effective and efficient anomaly detection models which can work in dynamic environments.

3. Anomaly Detection Models

Machine Learning has received an increase of attention due to its capabilities of solving business and societal issues. With correct preprocessing of data and tuning of training parameters, machine learning models can find an optimal solution for a given problem [9]. In this study, we chose the following Machine Learning models: eXtreme Gradient Boosting (XGBoost), Support Vector Machines (SVM), and Deep Convolutional Neural Networks (DCNN). We focused on the use of these machine learning models because they are well reputed for use in classification problems and they have been represented in numerous hackathons, competitions, and more [10].

3.1 Extreme Gradient Boosting

eXtreme Gradient Boosting or XGBoost [11] is a leading model for working with standard tabular data. It works under a gradient boosting framework used in regression, classification, ranking, and prediction. The model is built in a stage wise fashion and generalizes the prediction models or decision trees. Extreme gradient boosting differs from gradient boosting as it uses a more regularized model to help control overfitting [2]. This also aids in better performance. In a 2020 survey by Kaggle [12], XGBoost was in the top 3 most used methods and algorithms by employed data scientists as shown in Figure 1.

XGBoost involves three types of parameters: general, booster, and learning task. General parameters relate to the booster that is used. This is usually a tree or linear model. Booster parameters are chosen based on the type of booster that is used. Lastly, learning task parameters are chosen based on the learning scenario and differ depending on the given problem [2, 13].

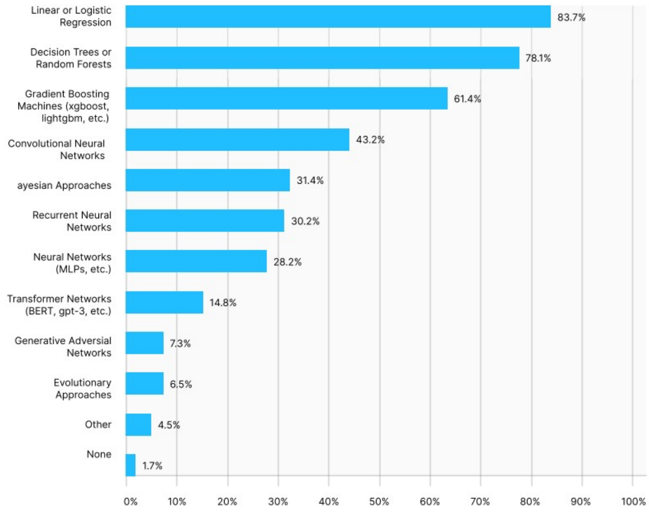

Fig. 1. Most common methods and algorithms used by Kaggle members who identify as employed "data scientist" [12].

3.2 Support Vector Machines

Support Vector Machine or SVM is a supervised algorithm used in regression and classification problems. It finds the best hyperplane that divides a dataset by classes. See Fig. 2. The further from the hyperplane the data lies, the more confident we are in their classification.

Generally, there is no exact single line that perfectly divides real data, and a curved boundary is used. We can use a "kernel trick" for non-linear inputs. The kernel is defined by $K(x,x^{'})$ where x and $x^{'}$ are the inputs, and K is the kernel used [15]. We have a variety of kernels to choose from, with the following as the most common:

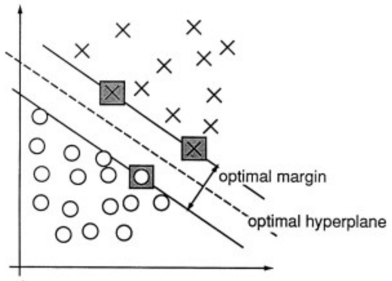

Fig. 2. An example of a separable problem in a 2-D space. The support vectors, marked with gray squares, define the margin of largest separation between the two classes [14].

Linear Kernel: the fastest but least accurate for multiple classes

$$K(x,x^{'}) = {}^{\mathrm{X}}\langle x,x^{'}\rangle \tag{1}$$

Polynomial Kernel: used for nonlinear models

$$K(x,x^{'}) = (\langle x,x^{'}\rangle +1)^{d} \tag{2}$$

where d is the degree of the polynomial. Radial Basis Function: most often used as it overcomes the space complexity issues of SVM

$$K(x,x') = \exp\frac{-||x-x'||^2}{2\sigma^2} \tag{3}$$

Sigmoid or Multi-layer Perceptron uses a single layer, preferred in neural networks

$$K(x,x^{'}) = \tanh(\rho\langle x,x^{'}\rangle + g) \tag{4}$$

where ρ is the slope and g is the intercept.

Common tuning parameters are C and Gamma. C is used for controlling the tradeoff between smooth decision boundary and classifying the training points correctly. Gamma defines how far a training example reaches [16].

3.3 Deep Convolutional Neural Network

Deep Convolutional Neural Networks (DCNNs) are models that can learn characteristics and classify them into labels. DCNNs consist of multiple layers and each layer performs specific tasks. The convolutional layer uses filters that activate when a feature is detected that will be learned. It then uses pooling layers that try to reduce overfitting. The fully connected layer is used for the actual classification of the network [17]. Table 1 shows the number of layers for each of the networks used. VGG16 and VGG19 are some of the oldest CNN (Convolution Neural Network) models.

Parameters used for DCNN vary, however some commonly used ones include MiniBatchSize, MaxEpochs, InitialLearnRate, ValidationFrequency, and ValidationPatience. A mini batch splits epochs into smaller epochs. MaxEpochs is the maximum epochs or iterations for the model to go through. InitialLearnRate is the learning rate where it controls how a model learns based on previous errors. ValidationFrequency validates data during training. Lastly, ValidationPatience stops training the data when the validation loss no longer decreases [2].

TABLE 1
NETWORK ARCHITECTURE

Network	Depth (layers)	Parameters (million)	Image Size
VGG16	16	138	224×224
VGG19	19	144	224×224
GoogLeNet	22	4	224×224
ResNet50	50	23	224×224
AlexNet	8	61	227×227

4. Experimental Consideration

XGBoost is designed for efficient processes and performance by implementing gradient boosted decision trees. The two main reasons that we chose to use XGBoost in our investigations are due to its execution speed, and model performance. Just like with the SVM algorithm, we tested the XGBoost algorithm with the NSL-KDD dataset using three different data splits. We split the data between training and testing with a split of 80% training and 20% testing, 75% training and 25% testing, and 65% training and 35% testing. The data set was composed of 125,974 captures and twenty-one classes with 27 features.

Deep Convolutional Neural Networks (DCNNs) are a framework that has shown groundbreaking success in many applications, including accurately classifying image data. For this investigation, we used the DCNN fusion architecture. Fusion allows for higher classification accuracy without increasing the number of layers in the neural network, especially when using suboptimal or incomplete data. First, the IoT data is converted from two-line element set (TLE) encoding to a vector. The data is then processed with one-hot encoding and converted to an image that can be fed into the DCNN. The features extracted from the penultimate layers from the DCNN implementations are combined and processed by dimension reduction algorithms, including t-distributed stochastic neighbor embedding (t-SNE) and principal component analysis (PCA). Finally, the features are clustered to classify the capture by the type of the attack that took place. This framework for fast and accurate classification can be applied to other datasets, such as satellite classification.

4.1 Preprocessing

During the preprocessing, we determined that the following features would be removed; as statistically they did not contribute to the training; or there were an overwhelming number of empty values.

- duration
- src bytes
- dst bytes
- land
- wrong fragment
- urgent
- hot
- count
- srv count
- serror rate
- srv serror rate
- same srv rate
- doff srv rate
- srv diff host rate

A combination of one-hot encoding for select remaining features and label encoding for our classes was used.

4.2 Training for XGBoost

The Python (3.10) libraries: Scikit-learn (version 1.1) and xgboost (version 1.5.0) were used to test the dataset. The MinMaxScaler function was used to scale the features. We transformed the array into a DMatrix format and entered in the desired tree booster parameters of max depth and eta. The learning task parameter multi:softmax was used in conjunction with the num_class parameter to classify the data. The evaluation metric merror was also calculated. In addition, root mean square error (RMSE) was evaluated to show how far predictions were from the correct values Since our data had multiple classifications, balanced accuracy was assessed.

4.3 Training for SVM

The data used for training SVM was the same preprocessed data used with XGBoost. Again, the data was standardized using MinMaxScaler. Different kernels were chosen, and the Linear kernel received the best overall performance. The regularization parameter was the value of C and the kernel coefficient for rbf and sigmoid kernals was gamma. Due to the number of classes, we used the one versus one strategy. This created extra computational time, and it was found we had to reduce the dataset or reduce the number of classes. Both options were explored.

The Support Vector Machine (SVM) algorithm has multiple kernel types such as Linear, Polynomial, RBF (Radial Basis Function), and Sigmoid kernels which were compared. By finding the most accurate kernel type to use for the SVM algorithm, it will create competitive results when comparing against the XGBoost and DCNN algorithms. This will give us an example of how the NSL-KDD dataset can be analyzed by multiple different algorithms, and the different results that each can give.

4.4 Training for DCNN

For the fully connected layers and the convolutional layer, the parameters Weight Learn Rate Factor and Bias Learn Rate Factor were used. For the fully connected layer, both were set to twenty. For the convolutional layer, they were set to ten. The training parameters were the execution environment, which was the GPU, Mini Batch Size of 10, Max Epochs set to 5, Initial Learn Rate of 0.0001, Validation Data using test set, Validation Frequency of 5, and Validation Patience of 10.

5. Results

Various metrics were calculated when running XGBoost, SVM, and DCNN. As Python was used for the SVM and XGBoost algorithms, the metrics were from scikit-learn functions. Matlab was used to run DCNN, and these metrics were calculated based on confusion matrices.

First, balanced accuracy was calculated. With data that is imbalanced, accuracy can be misleading. Using balanced accuracy is better in this case since it accounts for both the positive and negative outcome classes. F1 score was another metric taken into consideration. F1 score is the mean of precision and recall. It is a popular metric as it is a combination of two other important metrics. These metrics can be calculated as follows [17]:

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F1 Score} = 2\frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$

$$\text{TPR} = \frac{TP}{TN+FP}$$

$$\text{TNR} = \frac{TN}{TN+FP}$$

$$\text{Balanced Accuracy} = \frac{TPR+TNR}{2}$$

where TP is True Positive, FP is False Positive, TN is True Negative, FN is False Negative, TPR is True Positive Rate, and TNR is True Negative Rate.

Additionally, root mean square error (RMSE) was measured. This is used to evaluate the quality of predictions by showing how far predictions fall from true values. This is measured using Euclidean distance [2].

$$\text{RMSE} = \sqrt{\frac{\Sigma_{i=1}^{N} ||y_i - \hat{y_i}||^2}{N}} \tag{5}$$

The research results are detailed in the following.

5.1 XGBoost Results

Our experimental results show that the detecting performance scores of the XGBoost model depend on the training/testing split of the dataset. Both the 80/20 split (that is, 80% of the data used for training the model and 20% of the data used to test the detection effectiveness of the model) and the 65/35 split had scores where they excelled in while the 75/25 split was consistently either the lowest, or between the two scores. While the 65/35 split had the highest precision score of 89.3%, it continued to perform poorly through the other classifications when compared to the 80/20 split. As shown in Figure 3, the 80/20 split achieved a precision score of 84%, an accuracy score of 80.3%, an F1 score of 81%, and a recall score of 80.3%. This has led us to the conclusion that the 80/20 split was the split to use to better our investigation into the NSL-KDD dataset.

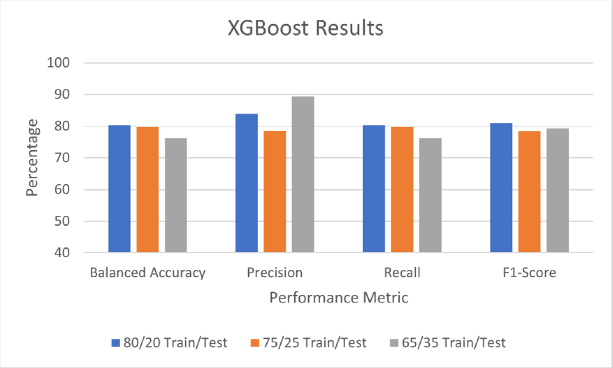

Fig. 3. XGBoost performance results using various splits

5.2 SVM Results

From our testing, we have found that the highest overall scores were received by the Linear kernel using a 75/25 split of the data between training and testing. The scores that it received were a 79.33% precision score, a 77.07% accuracy score, a 76.75% F1 score, and a 77.07% recall score. We can also see that the Linear kernel produced the best results for each split that was experimented with. For the 80/20 split, we received a precision score of 70.52%, an accuracy score of 77.06%, an F1 score of 71.76%, and a recall score of 77.06%. The 65/35 split gave a precision score of 67.57%, an accuracy score of 64.81%, an F1 score of 62.67%, and a recall score of 61.99%. Results shown in Figure 4 are using linear kernel as it achieved best results.

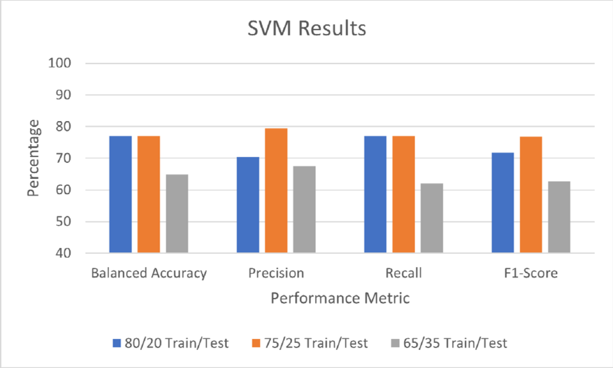

Fig. 4. SVM performance results using various splits

5.3 DCNN Results

Much like with the SVM and XGBoost algorithms, we have separated the NSL-KDD dataset into a few different training/testing splits including an 80/20 split, a 75/25 split, a 70/30 split, a 65/35 split, and a 60/40 split. Unlike with the other two algorithms we also split the results between a top-five accuracy classification and a top-one accuracy classification. For the top-five accuracy, the most accurate combination was a 70/30 split using the VGG19 neural network. For the top-one accuracy, the most accurate combination was a 65/35 split using the ResNet50 neural network. Both results will be taken into consideration when comparing the results received from the SVM and XGBoost algorithms.

5.4 Comparison of Results

Using the Precision, Accuracy, Recall, and F1-score metrics, we have now tested the XGBoost, SVM, and DCNN machine learning algorithms against the NSL-KDD dataset. After this testing, we will compare the results of each algorithm that used the most accurate configuration. For XGBoost, using an 80/20 split resulted in precision, accuracy, recall, and F1 scores of 84%, 80.7%, 81%, and 80.3%, respectively. The SVM algorithm using a 75/25 split, and the Linear kernel received precision, accuracy, recall, and F1 scores of 70.52%, 77.06%, 77.06%, and 71.76%, respectively. The DCNN algorithm using the Resnet50 framework with an 80/20 split resulted in precision, accuracy, recall, and F1 scores of 37.37%, 96.24%, 33.9%, and 35.55%, respectively. After obtaining all of these results we take a closer look at the differences between them. Although the DCNN algorithm resulted in an accuracy score of 96.24%, all other metrics paled in comparison with the next highest being precision at 37.37%. Compare this to both XGBoost and SVM where their biggest jumps were of around 4% and 7%, respectively.

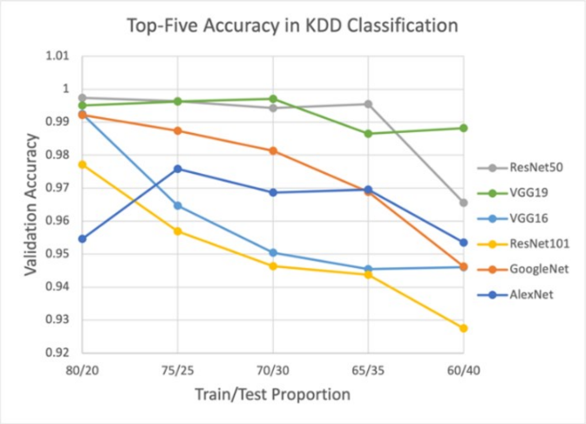

Fig. 5. Top-five accuracy classification results

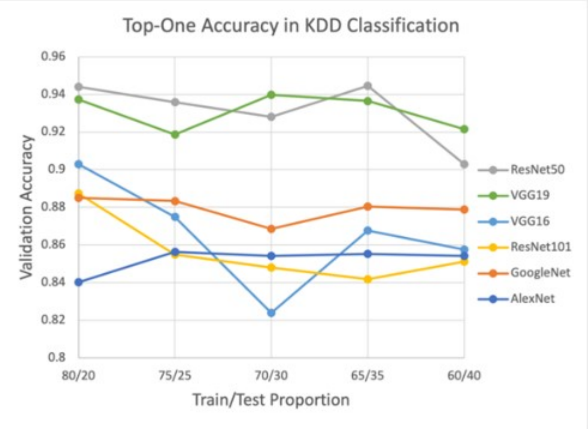

Fig. 6. Top-one accuracy classification results

We also looked at the efficiency of the detection (that is, how long it took to get these results). For both the XGBoost and SVM algorithms, it took only a matter of minutes, whereas to receive results for the DCNN algorithm it took a matter of hours. More specifically, the XGBoost algorithm was 425.53% faster than the SVM algorithm and 2,075.49% faster than the DCNN algorithm. From this comparison, we have found that even though the DCNN algorithm had the highest accuracy score of 96.24%, the XGBoost algorithm performed the best overall throughout this analysis.

6. Conclusion and Future Work

With the evolution of IoT devices and systems, protecting these devices and systems remains a significant and challenging issue in cybersecurity. As attacks on these devices and systems can be detrimental to humans and enterprises, it is of the utmost importance that IoT devices are protected. This study provides an analysis of how machine learning can be used to predict anomalies and this information can be used to prevent or mitigate future attacks. Machine learning algorithms used in this study included XGBoost, SVM, and DCNN. Each of these were fed data from the NSL-KDD dataset where the models classified anomalies and benign data. This study found that machine learning can effectively and efficiently be used for anomaly detection. Amongst the three algorithms, XGBoost proved to be the most effective and the most efficient method.

While the use of a dataset such as the NSL-KDD is useful for training and benchmarking the ML models, it does not directly compare to real world intrusion data. In collected real world data, there is less redundancy and more distinct types of attacks when compared to the dataset. In this study, the main point was to compare the accuracy and efficiency of the ML models that have been trained on the NSL-KDD dataset. The next step would be to collect and compile both anomalous and benign data created by various IoT devices from real world IoT applications. This data would then be used to create a new dataset to train and test the models. This could be compared to see how the model performances differ between real world data and data from the NSL-KDD dataset.

Acknowledgment

This work is supported by the NSA under NSA grant H98230-20-1-0296. We would like to thank Dr. Pearlstein, TCNJ, for the use of Zelda. In addition, we would like to thank all members of the IUP IoT Research team.

References

[1] S. Madakam, R. Ramaswamy, and S. Tripathi, “Internet of things (iot): A literature review,” *Journal of Computer and Communications*, vol. 3, pp. 164–173, 04 2015.

[2] M. Balega, W. Farag, S. Ezekiel, X.-W. Wu, A. Deak, and Z. Good, “IoT Anomaly Detection Using a Multitude of Machine Learning Algorithms,” *in the proceedings of the IEEE Applied Imagery Pattern Recognition Workshop,* Oct. 11-13, 2022, Washington, DC.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, 2009, pp. 1–6.

[4] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, “Evaluating intrusion detection systems: The 1998 darpa offline intrusion detection evaluation,” in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00*, vol. 2. IEEE, 2000, pp. 12–26.

[5] M. Palacios, “A Comparison of ANNs, SVMs & XGBoost on some Challenging Classification Problems,” Oct. 2019, eigenvector Research Incorporated. [Online]. Available: https://eigenvector.com/wp-content/ uploads/2020/03/Wise APACT Nonlinear Comparison.pdf

[6] S. Revathi and A. Malathi, “A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 12, pp. 1848–1853, 2013.

[7] J. Canedo and A. Skjellum, “Using machine learning to secure iot systems,” in 2016 14th Annual Conference on Privacy, Security and Trust (PST), 2016, pp. 219–222

[8] F. Hussain, R. Hussain, S. A. Hassan, and E. Hossain, “Machine learning in iot security: Current solutions and future challenges,” IEEE Communications Surveys & Tutorials, vol. 22, no. 3, pp. 1686–1721, 2020

[9] L. Wu and J. Fan, “Comparison of neuron-based, kernel-based, treebased and curve-based machine learning models for predicting daily reference evapotranspiration,” *PLOS ONE*, vol. 14, no. 5, pp. 1–27, 052019. [Online]. Available: https://doi.org/10.1371/journal.pone.0217520

[10] P. Das, et al., “Amazon sagemaker autopilot: a white box automl solution at scale,” *CoRR*, vol. abs/2012.08483, 2020. [Online]. Available: https://arxiv.org/abs/2012.08483

[11] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[12] “State of Machine Learning and Data Science 2020,” 2020. [Online]. Available: https://www.kaggle.com/kaggle-survey-2020

[13] Xgboost documentation. [Online]. Available: https://xgboost.readthedocs.io/en/stable/index.html

[14] C. Cortes and V. Vapnik, “Support-vector networks,” in *Machine Learning*, 1995, pp. 273–297.

[15] V. Jakkula, “Tutorial on support vector machine (svm),” *School of EECS, Washington State University*, vol. 37, no. 2.5, p. 3, 2006.

[16] R. Pupale, “Support vector machines(svm) – an overview,” June 2018. [Online]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989#:~:text=SVM

[17] S. Ezekiel, L. Pearlstein, A. Alshehri, A. Lutz, J. Zaunegger, and W. Farag, “Investigating gan and vae to train dcnn,” *International Journal of Machine Learning and Computing*, vol. 9, pp. 774–781, 12 2019.