Anomaly Detection Using Ensemble of Machine Learning Models

Md. Khairul Islam[1], Prithula Hridi[1], Md. Shohrab Hossain[1], Husnu S. Narman[2]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh
[2]Weisberg Division of Computer Science, Marshall University, Huntington, WV, USA
Email: khairulislamtanim@gmail.com, prithula5117@gmail.com, mshohrabhossain@cse.buet.ac.bd, narman@marshall.edu

***Abstract*—Anomaly detection system plays a significant role in recognizing intruders or suspicious activities inside the system, catching unseen and unknown attacks. Despite it has the weakness of having high false alarm rate, it can detect zero day attacks effectively. However benchmark anomaly detection datasets on modern traffic data are rare. In this paper, we have worked on a benchmark data set that reflects modern day network traffics. We have proposed an ensemble based machine learning for anomaly detection from the network traffic. Our results outperforms the state of the art model proposed by Moustofa et al. [1] on several metrics.**

***Index Terms*—anomaly detection, machine learning, network security.**

I. Introduction

As web applications are being increasingly popular, Internet has become a necessity in our day to day life. As a consequence, network systems are being targeted more by attackers with malicious intents. To detect intruders in network system, there are generally two approaches: signature based and anomaly based approaches. Signature based systems maintains database of previously known attacks and raise alarms if when any match is found with the analyzed data. However, they are vulnerable to zero-day attacks.

An anomaly in a network means deviation of traffic data from its normal pattern. Thus, anomaly detection techniques have the advantage of detecting zero-day attacks. However, in a complex and large network systems, it is not easy to define a set of valid requests or normal behavior of the end points. Hence, anomaly detection faces the disadvantage of having high false positive error rate (events erroneously classified as attacks).

There are different type of anomalies that can be mapped with different types of attacks. According to the study of Ahmed et al. [2], main attack types are DoS, Probe, U2R (User to Root) and R2U (Remote to User) attack. Ahmed et al. [2] mapped the point anomaly with the U2R & the R2U attacks, DoS attack to collective anomaly [3] and Probe attack to contextual anomaly.

Monowar et al. [4] has classified network anomaly detection methods and systems into six distinct classes. They are statistical, classification-based, clustering and outlier-based, soft computing, knowledge-based and combination learners.

In recent years, machine learning and deep learning, a branch of machine learning have become increasingly popular. They have been applied to different anomaly and intrusion detection systems [5] [6]. In many cases, they have outperformed the previous state of the art models. Because both of them have the potential to create better models, in this paper, we propose an ensemble of machine learning and deep learning based models for anomaly detection. To the best of our knowledge, this is the *first attempt* to explore both machine learning and deep learning extensively on the used dataset and to use a combination of both to outperform current state of the art work. Here lies the novelty of this work.

Our contributions in this work are as follows.

1) We have studied the performance of different machine learning and deep learning models on a benchmark anomaly detection dataset [7] for binary classification.
2) We have presented how varying learning rate, depth or other parameters impact on the performance metrics. We hyper-tuned each model to find out the best possible results for that model.
3) We have ensembled the predictions of the best performing models and presented a model which performs better than any of the models alone could. Then we compare the result of the final ensemble model with current state of the art.

Results show that our ensemble model outperforms the state of the art [1] in terms of detection rate and accuracy. It achieves 97.82% detection rate (whereas the previous best result was 92.70%). Our achieved accuracy is 93.69% (whereas previous best result was 93.40%).

The rest of the paper is organized as follows. In Section II we review the related researches in the field on network anomaly detection. Details about the dataset, methodology and performance metrics are described in Section III. Results of our approach are presented in Section IV. We have compared our work with state of the art in Section V. Finally, Section VI has the concluding remarks.

II. Relevant works

Eskin et al. [8] used the concept of the unsupervised SVM to detect anomalous events. Hu et al. [9] presented an anomaly detection method using the Robust SVM (RSVM). Yin et al. [10] used deep learning on NSL-KDD (2009) dataset.

Zhu et al. [11] used the attention-base Multi-Flow LSTM on CICIDS2017 dataset and achieved about 10% improvement in accuracy and recall on the multi-classification problems..

UNSW-NB15 is one of the benchmark datasets in anomaly detection. It was introduced by moustafa et al. [7]. The state of the art work on this dataset was done in [1], where the authors used a beta mixture model. Mixture models are probabilistic models for representing sub-populations within an overall population.

However, in recent times of machine learning and deep learning models have outperformed previous state of arts models on many occasions [12], [13], [11].

The state of the art work [1] on UNSW-NB15 dataset used Beta Mixture Model (a statistical and probabilistic model) to perform anomaly detection. Other models studied were also statistical ones. Machine learning and deep learning techniques have not yet been extensively applied to this dataset. Although the state of the art work on UNSW-NB15 has a low false alarm rate, its detection rate and accuracy can be further improved.

Therefore, in this paper, we have proposed an ensemble of machine learning and deep learning models, trained and tested on the UNSW-NB15 dataset [7].

III. Proposed Methodologies

In this section, we provide description about the dataset used, how it was pre-processed for using in model training. Then we explain the experimentation methods, how we trained and tested the models. Lastly, we provide the performance metrics used to evaluate and compare the results.

A. Dataset Description

For our study, we have used a popular benchmark dataset in NIDS research community, namely UNSW-NB15 [7] which is a hybrid of the real normal and modern synthesized attack of network data. It contains nine major families of attack types, such as fuzzers, analysis, backdoor, DoS, exploit, generic, reconnaissance, shellcode and worm. In [7], [14] the authors have discussed the benefits of this dataset compared to other benchmark datasets, such as KDD98, KDDCUP99 and NSLKDD.

Following the previous works, we have used the part of this dataset provided in [14] which is separated in train and test set. Since, the dataset is labelled, we have implemented machine learning algorithms for network anomaly detection in supervised learning. There are a total of 44 features. We have transformed all anomaly classes into a single class for binary classification. Table I shows the UNSW-NB15 dataset description.

TABLE I
UNSW-NB15 DATASET DESCRIPTION

Attack type	Train	Test
Normal	37000	56000
Anomaly	45322	119341
Total	**82322**	**175341**

B. Pre-processing

We have performed the preprocessing on the data set using the following three steps: droppoing unnnecessary columns, numericalization and scaling. Details are descibed as follows.

1) Dropping unnecessary columns: We have dropped ID columns from both train and test dataset as those can not be learned as features. Also, source and destination IP addresses and port numbers were excluded from the feature set.

2) Numericalization: As machine learning requires input data to be in numeric format, we have converted each categorical column into numeric by creating binary columns for each of their attributes. However, test set had 6 nominal values which never appeared in the train dataset. As they were absent in train data, model had no way of learning from them. So we have dropped the binary columns for these labels. Finally, we had 188 features in our train data set.

3) Scaling: We have applied standard scaling using StandardScaler from sklearn's preprocessing library. This scaler calculates the mean and standard deviation of training samples and converts them using the following equation:

$$x = \frac{x - \mu}{\sigma} \tag{1}$$

where, μ is the mean value and σ is the standard deviation.

C. Methodology

For our machine learning models, we have used sklearn library of python and for neural network, we used Keras [1], a python deep learning library developed by Google. We used kaggle [2] kernels with GPU enabled to run our models.

As the dataset comes with separate train and test set, we have trained the models on train data and varied different parameters to show the impact on the test data. For each model, we have presented the baseline parameters, parameters used in hyper-tuning and how varying parameters impacted the classification results. Then finally we ensemble the predictions of the best performing models and give the final performance result in Section IV-F. Models used in ensemble are RandomForest, GradientBoosting, XGB, LightGBM, Neural Network(dense layer based).

D. Evaluation metrics

Here, we discuss all The performance metrics used to compare the performance of our approach with previous works.

- **True Positives (TP)** : The cases in which we predicted YES and the actual output was also YES.
- **True Negatives (TN** : The cases in which we predicted NO and the actual output was NO.
- **False Positives (FP)** : The cases in which we predicted YES and the actual output was NO.
- **False Negatives (FN)** : The cases in which we predicted NO and the actual output was YES.

[1]https://keras.io/
[2]http://kaggle.com

1) Accuracy: It is the ratio of number of correct predictions to the total number of input samples

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

2) Precision: It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$precision = \frac{TP}{TP + FP} \quad (3)$$

3) Recall or DR(Detection Rate): It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive)

$$recall = \frac{TP}{TP + FN} \quad (4)$$

4) FPR: The false positive rate (FPR) is the proportion of incorrectly identified observations, that is

$$FPR = \frac{FP}{FP + TN} \quad (5)$$

5) F1-score: F1-score is the harmonic Mean between precision and recall.

$$f1\_score = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \quad (6)$$

IV. Experiment and Results

In this section, we describe experimentation methods and results of various machine learning based models on unsw-nb15 dataset. In each part, we have described the hypertuning part of a model and the best possible results from it. The best results are marked using * in each table.

A. RandomForestClassifier

We have used RandomForestClassifier from sklearn.ensemble. The baseline parameters are listed in Table II. We set prediction threshold at 0.4 for this model. From Table III, we can see that changing max depth did not have much effect on the model performance except reducing it a bit. However, when we increase the number of estimators beyond 100, the accuracy decreases. This is because the model then overfits on train data.

TABLE II
RandomForest baseline parameters

Parameter	Value	Parameter	Value
min_samples_leaf	1	class_weight	None
max_features	auto	max_leaf_nodes	None
bootstrap	True	random_state	0

Also, decreasing the estimators less than 100 has negative impact on the performance. The best model was achieved with 100 estimators and max depth at 50 (see Table III).

TABLE III
RandomForest classifier results

n-estimators	depth	accuracy	f1-score
50	50	93.24	95.22
50	150	93.33	95.27
100	50	93.39*	95.31*
100	150	93.33	95.27
200	50	93.36	95.30
200	150	93.34	95.29
500	50	93.03	95.31
500	150	93.01	95.29

$*$ best parameters

B. GradientBoosting

We have used GradientBoostingClassifier from sklearn.ensemble. We have varied n_estimators and learning rate for this model. Each estimator has regression trees with maximum depth 3. We set prediction threshold at 0.5 for this model. The baseline parameters of this classifier are listed in Table IV. From Table V, we find that with the increase the number of estimators, the accuracy of the classifier is increased.

TABLE IV
GradientBoosting baseline parameters

parameter	value	parameter	value
learning_rate	0.1	presort	auto
n_estimators	100	max_depth	3
random_state	0	max_features	None

We did not increase it further to prevent overfitting. Also, decreasing learning rate than 0.1 had negative impact on performance, because the model was learning too slow to converge.

TABLE V
GradientBoosting Classifier results

n-estimators	learning rate	accuracy	f1-score
100	0.05	89.82	92.31
100	0.1	92.09	94.12
500	0.05	92.40	94.49
500	0.1	92.61*	94.67*

$*$ best parameters

C. LightGBM

LightGBM is a gradient boosting framework from Microsoft that uses tree based learning algorithms.

TABLE VI
LIGHTGBM BASELINE PARAMETERS

parameter	value	parameter	value
bagging_frequency	5	base_score	0.9
early_stopping_round	100	boost	gbdt
num_boost_round	2000	max_bin	255
objective	binary	num_leaves	31

The baseline parameters for the LightGBM model are listed in Table VI. We have varied four parameters of this model: learning rate, max-depth, bagging fraction, feature fraction. We set prediction threshold at 0.4 for this model. From Table VII we find that when the learning rate is decreased, the performance (accuracy) of the model also decreased. This is because it fails to convergence with a slower learning rate. We found setting constraint on depth gives improved results. Among them, best results was found at depth 10. Also, adding fraction less than 1.0 improved classifier performance, this fraction means what fraction of the dataset and features would be used for training in the next iteration.

TABLE VII
LIGHTGBM RESULTS

learning rate	max depth	bagging and feature fraction	accu-racy	f1-score
0.1	10	0.5	92.69	94.71
0.1	10	1	92.75*	94.78*
0.1	15	0.5	92.63	94.69
0.1	15	1	92.59	94.67
0.01	10	0.5	92.75	94.69
0.01	10	1	92.55	94.59
0.01	15	0.5	92.84	94.76
0.01	15	1	92.59	94.62

$*$ best parameters

D. *XGBClassifier*

XGBoost is an optimized distributed gradient boosting library. We have worked on the XGBClassifier from XGBoost. We have replaced the base score with 0.9, as it improved the performance. Prediction threshold was set to 0.2. From Table VIII, we can see that performance decreases randomly with the increase of number of estimators. However, the result improved for higher learning rate. But increasing learning rate too much can made the model fail to converge.

TABLE VIII
XGB CLASSIFIER RESULTS

n-estimators	learning rate	accuracy	f1-score
100	0.1	91.45	93.71
100	0.5	92.13*	94.34*
500	0.1	91.94	94.23
500	0.5	90.42	93.3

$*$ best parameters

E. *Artificial Neural Network*

We used Keras framework for implementing neural network. We tried different combination of layers, units of dense and dropout layer. Dense layer is a densely connected neural network layer, which connects every input to every cell unit. Dropout layer randomly sets a fraction of input units to 0, reducing overfit.

Our number of epochs was 10 and we took the best model based on validation accuracy. From our results (shown in Table IX), we can see that increasing input layer size increases model performance, because it helps to learn more from the input data. However, using more layers caused the model to overfit. Also adding dropout layer did not solve the issue. So, the best model was a neural network with 512 dense input units, then 32 dense input units, following the final output unit. All dense layers except the last one used activation method 'relu'. The last layer had activation method 'sigmoid'.

TABLE IX
ANN RESULTS

Model architecture	accu-racy	f1-score
dense(256)dense(32)dense(1)	91.97	94.01
dense(256)drop(.2)dense(32)dense(1)	91.84	93.86
dense(256)dense(64)dense(32)dense(1)	91.82	93.84
dense(512)dense(32)dense(1)	92.05*	94.06*

$*$ best architecture

F. *Ensemble*

Ensemble method is a machine learning technique that combines several base models in order to produce one optimal predictive model. In this part, we have taken our best performing models and used their predictions to give the final output. For each model, the best performing one is marked by * sign.

There are several ways ensemble can be used. We took a simple approach called majority voting. It works by taking prediction from everyone and giving the output in favour of the class for which most of the models gave same output. After applying voting on the predictions of all models on test data our final result reached 93.69% accuracy, 93.24% precision, 97.82% recall and 95.48% f1-score on the test dataset.

V. Comparison with state of the art

Our final model achieves 93.69% accuracy, 93.24% precision, 97.82% recall and 95.48% f1-score on the separate test dataset. Our ROC-AUC score is 98.45% which is also very high.

Compared to previous works, our model outperforms the state of the art model by Moustafa et al. [1] as we can see from table X. However, our FPR is 15.11%, which is higher than theirs, so our false alarm rate is higher than their work.

TABLE X
Comparison with state of the art

Metrics	Moustafa *et al.* [1]	Our Model
DR(%)	92.70	97.82*
Accuracy(%)	93.40	93.69*
FPR(%)	5.90*	15.11

$*$ better result

Our ensemble model has a greater detection rate which is possible as even if one model fails to detect an anomaly, there are other models which does not fail. Thus, this ensemble version detects more anomaly connections, thereby making our model better in terms of detection rate. However, as the ensemble model detects more connections as anomaly, it also makes more false alarms (meaning normal connections as anomaly). That is why our false positive rate is higher than the previous work.

As our models ensembles several models, some of which takes much time to train. Hence, our model is slower to learn compared to previous statistical based model. Nevertheless, for sensitive connection and for better detection, our model is certainly the better choice.

VI. Conclusion

In this paper, we have worked on a benchmark data set and proposed an ensemble based machine learning for anomaly detection from the network traffic. Our ensemble of machine learning models, trained on UNSW-NB15 anomaly detection dataset, outperforms tne current state of the art in terms of detection rate and accuracy. we have clearly shown how machine learning can be used on this dataset to surpass traditional probabilistic models.

Our ensemble model has a greater detection rate which is because if one model fails to detect an anomaly, there are other models which does detect the anomaly. Thus, this ensemble version detects more anomalous connections, thereby making our model better in terms of detection rate.

References

[1] N. Moustafa, G. Creech, and J. Slay, "Anomaly detection system using beta mixture models and outlier detection," in *Progress in Computing, Analytics and Networking*. Springer, 2018, pp. 125–135.

[2] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[3] M. Ahmed and A. N. Mahmood, "Network traffic analysis based on collective anomaly detection," in *9th IEEE Conference on Industrial Electronics and Applications*. Hangzhou, China: IEEE, 9-11 June, 2014, pp. 1141–1146.

[4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, First 2014.

[5] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10586-017-1117-8

[6] P. Casas, P. Fiadino, and A. D'Alconzo, "Machine-learning based approaches for anomaly detection and classification in cellular networks." in *TMA*, 2016.

[7] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Military communications and information systems conference (MilCIS)*. Canberra, Australia: IEEE, 10-12 Nov. 2015, pp. 1–6.

[8] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[9] W. Hu, Y. Liao, and V. R. Vemuri, "Robust anomaly detection using support vector machines," in *International conference on machine learning*, 2003, pp. 282–289.

[10] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[11] M. Zhu, K. Ye, Y. Wang, and C.-Z. Xu, "A deep learning approach for network anomaly detection based on amf-lstm," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2018, pp. 137–141.

[12] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, Feb 2018.

[13] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Oct 2016, pp. 258–263.

[14] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.