Network intrusion detection: a comparative study of four classifiers using the NSL-KDD and KDD'99 datasets

Ananya Devarakonda[1], Nilesh Sharma[1], Prita Saha[1] and Ramya S[2]

[1] Student, Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India

[2] Associate Professor, Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India

E-mail: ramya.lokesh@manipal.edu

Abstract. As most of the population acquires access to the internet, protecting online identity from threats of confidentiality, integrity, and accessibility becomes an increasingly important problem to tackle. By definition, a network intrusion detection system (IDS) helps pinpoint and identify anomalous network traffic to bring forward and classify suspicious activity. It is a fundamental part of network security and provides the first line of defense against a potential attack by alerting an administrator or appropriate personnel of possible malicious network activity. Several academic publications propose various artificial intelligence (AI) methods for an accurate network intrusion detection system (IDS). This paper outlines and compares four AI methods to train two benchmark datasets- the KDD'99 and the NSL-KDD. Apart from model selection, data preprocessing plays a vital role in contributing to accurate solutions, and thus, we propose a simple yet effective data preprocessing method. We also evaluate and compare the accuracy and performance of four popular models- decision tree (DT), multi-layer perceptron (MLP), random forest (RF), and a stacked autoencoder (SAE) model. Of the four methods, the random forest classifier showed the most consistent and accurate results.

Keywords: Network intrusion detection, network security, KDD'99, NSL-KDD, deep learning, random forest, decision tree, autoencoder

1. Introduction

Network traffic anomaly detection is becoming an increasingly demanding task as more people get access to the internet. According to [1], about two-thirds of the globe will be connected online by the beginning of 2023, estimated to be around 29.3 billion networked devices. It is thus essential to assure the safety and privacy of users online by swiftly detecting and reporting malicious network activity. Briefly, network intrusion refers to potentially malicious unauthorized activity on a digital network. An intrusion can likely threaten the confidentiality, integrity, and accessibility of a computer network, causing issues ranging from a breach of privacy to compromising systems. Examples of network attacks include denial-of-service (DoS), asymmetric routing, buffer overflow attacks, traffic flooding, spoofing, trojans, and worms.

According to [2], the main types of intrusion detection systems are: signature based detection systems and anomaly based detection systems. Signature based systems primarily rely on known attack signatures to detect unauthorized activity. In contrast, in situations where

attack signatures are unknown, detection of anomalous network activity is mainly carried out by anomaly based systems. Such detection systems use various methods ranging from statistical to deep learning models to detect suspicious network activity automatically. This paper focuses on models for an anomaly based IDS trained on the KDD'99 [3], and the more recent NSL-KDD datasets [4].

Contributions of the paper include:

(i) Proposing an effective data preprocessing strategy that attempts to accurately represent the features in the dataset to reduce model bias.

(ii) Introducing and comparing the use of four models in the application of an IDS- decision tree (DT), multi-layer perceptron (MLP), random forest (RF), and a stacked autoencoder model (SAE).

(iii) Comprehensively listing essential features from the two datasets using the most accurate of the four models discussed.

2. Relevant work

The Knowledge Discovery and Data Mining (KDD) Cup conducted in 1999 featured the KDD'99 dataset [3], a subset of the Defense Advanced Research Projects Agency (DARPA) 1998 benchmark dataset. Since the contest, researchers have extensively used the dataset to train and test AI models for an accurate IDS. This section discusses relevant work in modeling intrusion detection systems using machine learning (ML) and deep learning (DL) solutions on the NSL-KDD [4] and KDD'99 datasets- the former being an improved variant of the KDD'99. Mukkamala et al. [5] presented a neural network (NN) and a support vector machine (SVM) to perform binary (attack/ normal) classification. They also extracted thirteen important features, trained the two models using these extracted features, and reported the results. They concluded that both SVMs and neural networks provided accurate results, with the SVMs performing slightly better. Further, the models trained on the thirteen extracted features showed only slightly lowered accuracy.

Chan et al. [6] compared fusion approaches- majority vote, Naïve Bayes combination, Dempster-Shafer combination, average, and neural networks on the KDD'99 dataset. They only used the training set from the KDD'99 dataset and classified amongst ordinary traffic and six selected denial-of-service (DoS) attacks. They concluded that the radial basis function neural network (RBFNN) fusion approach outperformed the other fusion methods.

Heba et al. [7] used PCA to choose attributes in the dataset and reduce the dimensionality of the said attributes. The selected features were then passed to an SVM to conduct 5-class classification- normal, remote-to-local (R2L), DoS, probe, and user-to-root (U2R). Gao et al. [8] introduced a deep belief network (DBN) for an IDS trained on the KDD'99 dataset and concluded that they performed better than SVMs and artificial neural networks (ANNs).

More recently, [9] introduced a hybrid deep neural network model to conduct binary and five-class classification. They reported model performance on various datasets that included the KDD'99 and NSL-KDD. Liu et al. [10] used the dataset NSL-KDD on a hybrid model consisting of K-means, RF, and DL components. They achieved an 85.24% accuracy on the dataset. Another hybrid model discussed by Wang et al. [11] described an SDAE-ELM. Their aim was to tackle a few limitations of DL models such as the long time required to train such models. They demonstrated that the model performed well compared to other ML methods. Shone et al. [12] presented an IDS with a non-symmetric deep autoencoder (NDAE) and an RF classifier. They used the 10% subset from the KDD'99 and used the entire NSL-KDD dataset for evaluations. They reported results on the former dataset for five-class classification and the latter dataset for five-class and thirteen-class classification. They obtained accurate results and

provided an efficient solution that reduces training time by up to 98.81%. Other methods that use deep learning for intrusion detection include [13–16].

A popular method of classification is the use of an RF model. Ahmad et al. [17] compared SVM models, RF models, and extreme learning machines (ELMs) for an effective IDS using data from the dataset NSL-KDD. They concluded that ELMs outperform other methods. It is essential to keep in mind that the authors excluded non-numeric features while preprocessing the dataset. Negandhi et al. [18] selected important features from the NSL-KDD dataset using Gini importance to train a random forest classifier. They obtained results with a 99.88% accuracy.

3. Datasets used

3.1. KDD'99 dataset

As mentioned previously, the KDD'99 dataset [3] is a subset of the Defense Advanced Research Projects Agency (DARPA) 1998 benchmark dataset. The dataset was used for the Third International Knowledge Discovery and Data Mining (KDD) Tools Competition. The training and test data contain forty-one features. However, the training and test dataset labels are from different distributions.

The training data contains twenty-four labels categorized as normal and twenty-three types of attack. The test data contains fourteen more attacks. The attack types belong to one of the given categories: U2R, DoS, R2L, and probing attack. The dataset attributes are continuous, discrete, or symbolic. It is pertinent to keep in mind that most features are imbalanced. For reference, table 1 illustrates the frequency of outcome labels in the dataset by presenting high, mid, and low frequency attributes. Clearly, the "smurf." attack label occurs significantly more frequently than the "spy." attack label that occurs only two times in the dataset. The entire dataset contains around 743MB of uncompressed data. We used the entire dataset to train the models and verify the results.

3.2. NSL-KDD dataset

Tavallaee et al. [4] outlined several limitations of the KDD'99 dataset. They thus proposed the NSL-KDD that distilled the original KDD'99 dataset with specific improvements. They removed redundant and duplicate records present in the KDD'99 dataset. The authors also took steps to avoid model bias by making the number of records inversely proportional to their success of prediction. The entire official dataset is split into a training dataset containing 125,973 records along with a test dataset containing 22,544 records. The dataset contains the forty-one original features of the KDD'99 dataset. Table 1 provides a comparative study of the frequency of outcome labels in the dataset.

Table 1: Frequency of outcome labels

KDD'99		**NSL-KDD**	
Outcome	Count	Outcome	Count
smurf.	280790	normal	77054
neptune.	107201	neptune	45871
teardrop.	979	snmpgetattack	178
pod.	264	httptunnel	133
perl.	3	sqlattack	2
spy.	2	spy	2

3.3. Limitations of the datasets

The DARPA 1998 intrusion detection and evaluation dataset received several criticisms [4, 19–21]. As the KDD'99 dataset (and therefore, the NSL-KDD dataset) contains data from the DARPA 1998 dataset, it is relevant and pertinent to discuss observations from the DARPA 1998 dataset [3]. This characteristic of the KDD'99 dataset leads to error propagation, and many criticisms of the DARPA 1998 dataset are valid for the datasets used in the paper. The main critique of the DARPA 1998 dataset is that it did not model real-world network traffic and the contents of the dataset often led to over-optimistic results due to model bias.

For instance, [19] used Snort against the DARPA 1998 dataset and found that it performed poorly, having low accuracy of detection and high false positivity rates. They concluded that the low accuracy of results could be attributed to the dataset not modeling attacks correctly or that it was simply outdated. They further mentioned that several advanced intrusion detection systems evaluated using the DARPA 1998 data accurately predicted DoS and probe attacks (as they frequently occurred in the dataset) but struggled with predicting U2R and R2L attacks. On the other hand, Snort detected U2R and R2L attacks with ease but struggled with accurately detecting DoS and probe attacks. Furthermore, [20] critiqued comparative evaluations of intrusion detection systems conducted by the Lincoln Laboratory of MIT funded by DARPA in 1998 as well as 1999 and pointed out questionable evaluation methods leading to potentially biased results. [21] pointed out that the train and test sets from the KDD'99 dataset represented dissimilar target hypotheses for the R2L and U2R outcome labels, leading to low accuracy in the misuse detection of these categories.

Apart from these fundamental issues, [4] addressed problems such as redundant records, data imbalance, and attributes of the KDD'99 dataset that could lead to model bias. They proposed that although the NSL-KDD dataset is not a real-world representation of network traffic today, it could serve as a benchmark dataset.

It is essential to address that releasing open-source datasets in this field without risking privacy issues is challenging. On the one hand, simulated data may not accurately represent real-time network traffic, and on the other, open-source datasets with real-world data contain heavily anonymized values, leading to other modeling issues. It is important to remember that despite these limitations, the KDD'99 and NSL-KDD datasets have popularised research in modeling anomaly-based intrusion detection systems as they were one of the earliest open-source datasets containing network and attack data. They are, to date, widely used benchmark datasets to evaluate models for IDS applications.

4. Methodology

4.1. Data preprocessing

Both the datasets used in the paper contain a mixture of continuous, discrete, and symbolic data. The steps provided in this section apply to both datasets.

Neither dataset has a header row. For convenience, before preprocessing, referring to the official documentation [3, 4], a header row with feature names is added to both datasets. An exploratory analysis of the datasets revealed that the feature num_outbound_cmds only contains zeros. Thus, due to the feature being redundant, num_outbound_cmds is dropped. The features are then split into numeric (continuous) and categorical (symbolic or binary) types. Continuous numeric features such as duration (duration of connection) and src_bytes (data bytes) present in the dataset are normalized using z-score normalization.

Features such as protocol_type (specifies the type of protocol, such as udp and tcp) and service (contains the name of services such as http and telnet) contain nominal categorical variables. Thus, these features are one-hot encoded to retain their intrinsic quality of not being in a particular order. Formally, the procedure for one-hot encoding is as follows:

Considering that X is a feature with discrete categorical variables $x_1, x_2, ..., x_n$ where

$x_1, x_2, ..., x_n$ represent distinct values, the one-hot encoding of x_i is represented as a vector u. Every value of u is equated zero except the ith value. The ith component of the vector u contains the value one. To illustrate this, say $X = \{x_1, x_2, x_3\}$ (here $n = 3$) then after one-hot encoding, x_1 is $[1, 0, 0]$, x_2 is $[0, 1, 0]$ and x_3 is $[0, 0, 1]$.

Binary features contained in the dataset such as, logged_in (a binary feature that mentions the success of login), are retained as they are.

For the MLP and SAE models, the entirety of the KDD'99 dataset is split so that 90% of the data is reserved for the training step, whereas the rest is used for testing. The training dataset is then further split so that 80% of it is used to train the MLP and SAE. The remaining 20% is used to validate the models. For the RF and DT models, we use 99% of the KDD'99 dataset for training and the remaining 1% to test the classifiers. The NSL-KDD dataset is split in the same way as described above for all four models.

4.2. Models

This paper examines a total of four models- two deep learning models (MLP and SAE models) and two machine learning models (RF and DT classifiers). This section provides a detailed description of the models.

4.2.1. Multi-layer perceptron The first deep learning method we explore is the MLP illustrated in figure 1(a). Fundamentally, an MLP contains the following layers: one input, n hidden (where $n \geq 1$), and one output. The depth of the MLP is determined by n, the number of hidden layers. The value of n is chosen by analyzing the training loss, validation loss, and the final accuracy of the model. We choose the model with the greatest accuracy. Mathematically, each hidden unit is represented as:

$$h_i^{(l)}(x) = f^{(l)}\Big(\sum_j w_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)}\Big) \tag{1}$$

Where, $h_i^{(l)}$: represents the ith hidden unit present in the lth layer

$f^{(l)}$: is the non-linear activation for the lth layer

$w_{ij}^{(l)}$: refers to the weights for the lth hidden layer

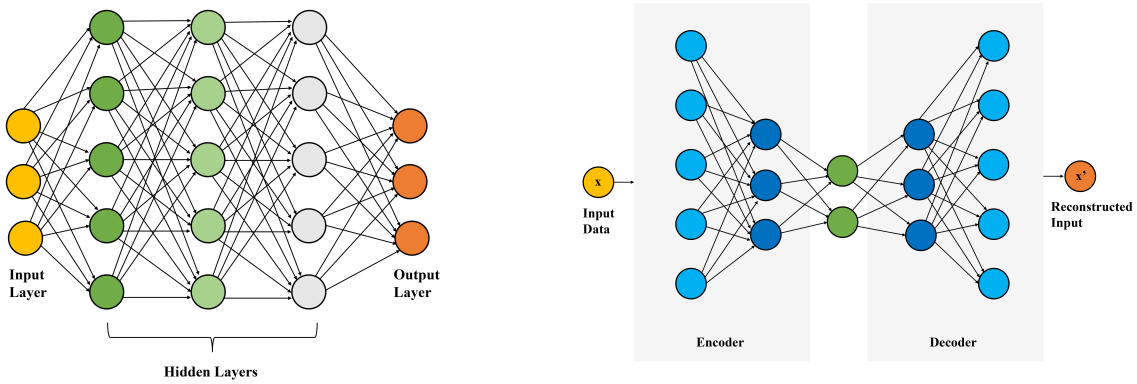
$h_j^{(l-1)}$: output of units from the previous hidden layer

$b_i^{(l)}$: refers to the bias terms for the lth hidden layer

The model used to train the KDD'99, and NSL-KDD dataset utilized a total of three hidden layers. The first and third hidden layers had fifteen nodes, while the second had a hundred nodes. All layers (not including the output layer) used ReLu as the activation function. As the task is to perform categorical classification, the softmax function, given in equation (2), was used as activation for the output layer.

$$\sigma(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2}$$

The model used categorical cross-entropy loss and Adam optimization with default values [22]. The training was done for fifty epochs with early stopping to avoid over-fitting.



(a) Neural network model

(b) Stacked autoencoder model

Figure 1: An illustration of neural networks and stacked autoencoder models

4.2.2. Stacked autoencoder (SAE) An autoencoder (AE) is a popular self-supervised deep learning model that is trained to generate an output similar to that of the input. The architecture of an autoencoder is similar to that of the neural network model; however, unlike the traditional neural network, the autoencoder features a bottleneck layer. This bottleneck layer attempts to learn lower-dimensional representations of features in the dataset. Thus, the autoencoder contains an encoder to encode latent representations of features in the dataset and a decoder that reconstructs the given input. If a sparsity constraint is used on the hidden units of the autoencoder model, the autoencoder is called a sparse autoencoder.

An SAE is a model that consists of numerous sparse AEs, as shown in figure 1(b). SAEs are known to show promising results as they consider spatial and temporal correlations [23]. It is essential to note that SAEs do not require labelled data, being a self-supervised learning method, and thus, they can easily handle mislabelled data.

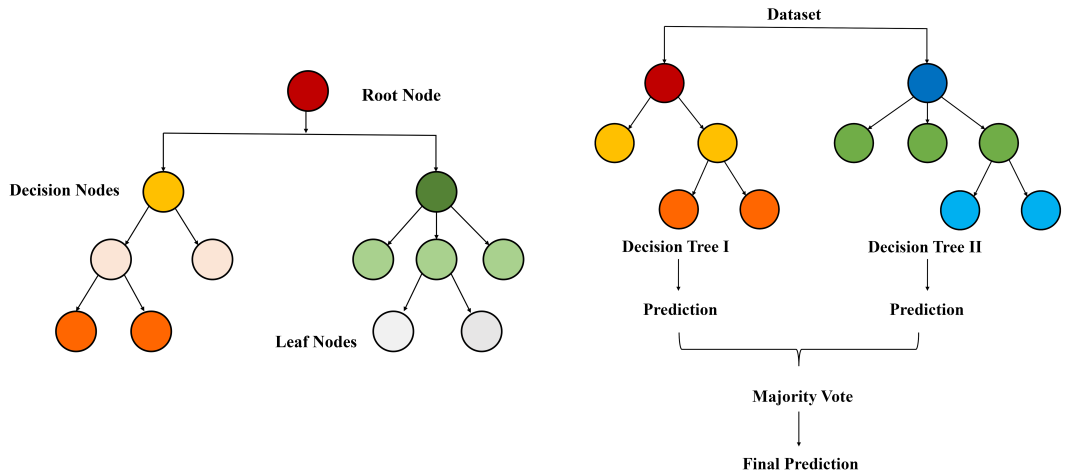
For this project, we use an SAE model to learn to emulate network traffic data under the "normal" outcome label. The fundamental idea is that having learned to reproduce normal network traffic, such an SAE model will have a tough time emulating internet traffic representing an ongoing attack. The results section of the paper goes further in detail about this property of the autoencoder.

The SAE model used contained three hidden layers where the first and third layers contained thirty hidden units, while the second hidden layer (bottleneck layer) contained five hidden units. The model used mean squared error as loss and Adam optimization. Similar to that of the neural network model, early stopping was used to prevent over-fitting.

4.2.3. Decision tree classifier A DT can be used for classification by recursively partitioning features in the dataset. A decision tree classifier uses a flow-chart or tree-like structure where each node represents a feature in the dataset and tree partitions are based on the attribute values, as depicted in figure 2(a). For this paper, attribute order in the decision tree is chosen based on Gini impurity. The Gini impurity is a measure of noise present in the features in the dataset. Nodes that are higher in the decision tree hierarchy have a lower Gini impurity and vice-versa. The formula to compute the Gini impurity is:

$$Gini(p) = 1 - \sum_{i=1}^{N} p_i^2 \tag{3}$$

Where p_i represents the probability that an item is labelled with class i in the dataset among N classes.



(a) Decision tree classifier model (b) Random forest classifier model

Figure 2: An illustration of decision tree and random forest classifiers

4.2.4. Random forest classifier The RF classifier [24], depicted in figure 2(b), is a supervised machine learning model that uses the theory behind the decision tree classifiers. Briefly, the random forest classifier constructs several decision trees based on random subsets of features from bootstrapped datasets.

The decision trees construct nodes based on Gini impurity (equation (3)), and the predicted output of the RF is made based on the majority vote of predictions from these decision trees. One significant advantage of random forest classifiers over decision tree classifiers is that they are less prone to over-fitting being an ensemble method. They are also known to be unaffected by imbalance and sparsity in the dataset [25].

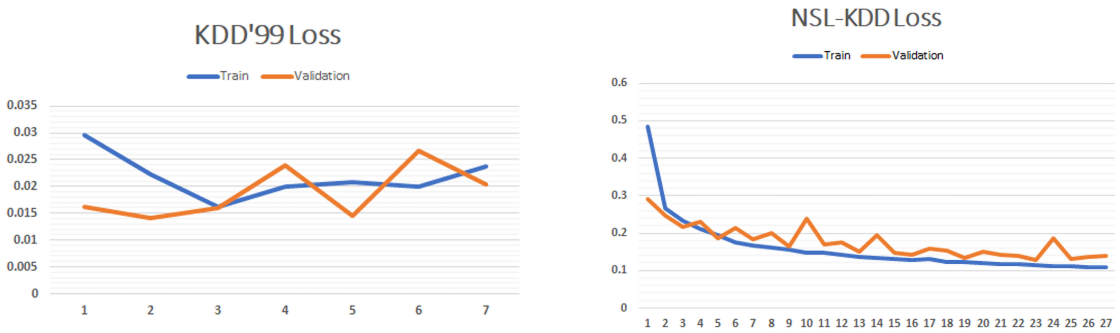
4.3. Software

For this project, we use Python 3.9.2 [26] and the following libraries- Pandas [27] and Numpy to preprocess the KDD'99 and NSL-KDD datasets. The final deep learning models were implemented using Tensorflow2.0 [29]. The DT and RF classifiers were implemented using Scikit-Learn [30].

5. Results

This section reports results based on the accuracy metrics mean squared error (MSE), root mean squared error (RMSE), precision, F1-score, and recall. Metrics, MSE and RMSE can sometimes be misleading when discussing results from models trained on an imbalanced dataset. Thus, to ensure that the model is accurate in predicting all classes, precision, F1-score and recall are also used to evaluate the model.

As the SAE model worked differently from the rest, results from the SAE model are specified separately. When trained on the KDD'99 dataset, for the data classified as regular internet traffic, the SAE had a validation MSE of 0.058 and a test MSE of 0.182. For data classified



(a) KDD'99 train and validation loss

(b) NSL-KDD train and validation loss

Figure 3: Training and validation loss over epochs of training on the two datasets

Table 2: Accuracy of predictions from various models

Dataset	Model Name	Accuracy Metric (10^{-2})				
		MSE	RMSE	Precision	Recall	F1-Score
NSL-KDD	NN	381.6	195.36	97.5087	97.015	96.2636
	RF	**11.26**	**33.57**	**99.7884**	**99.761**	**99.6897**
	DT	34.92	59.093	99.3841	99.285	99.2109
KDD'99	NN	9.047	30.078	99.8529	99.82	99.7553
	RF	0.714	8.4528	**99.9939**	**99.993**	**99.9928**
	DT	**0.249**	**4.9905**	99.9918	99.991	99.9918

as an attack, the SAE had a much higher MSE of 0.353. When the NSL-KDD was used for training the SAE model, the SAE reconstructed normal traffic with a validation MSE of 0.029 and a test MSE of 0.093. For attack traffic, the SAE once again had a much higher MSE of 0.274. Using the fact that the MSE significantly increases when the data represents an attack, traffic data can be classified as normal or abnormal. Such an approach is convoluted and only provides binary (attack/normal) predictions.

Looking at the results from the other models listed in table 2, we observe that the DT classifier has the lowest MSE and RMSE for the KDD'99 dataset, whereas the RF classifier has the highest precision, F1-score, and recall. Thus, for the KDD'99 dataset, the best model to use is the RF classifier. A similar trend is observed for the NSL-KDD dataset, where the random forest classifier provides the most accurate results. Along with the results mentioned in table 2, for the MLP model, figure 3 displays the loss curves obtained while training both datasets.

As the random forest classifier shows the most accurate results, we have used it to extract important features from both datasets for further analysis. For both datasets, count, diff_srv_rate, and dst_bytes are found to be the top three features. Other important features identified are protocol_type and service_type.

6. Discussion

The results clearly show that the random forest classifier is the most accurate amongst the four models used. This is because of the fact that the random forest classifier is mostly impervious to an imbalance in the dataset, and according to [25] it adapts well to sparsity. The difference in accuracy between the two datasets can be accounted for the fact that attributes in the KDD'99

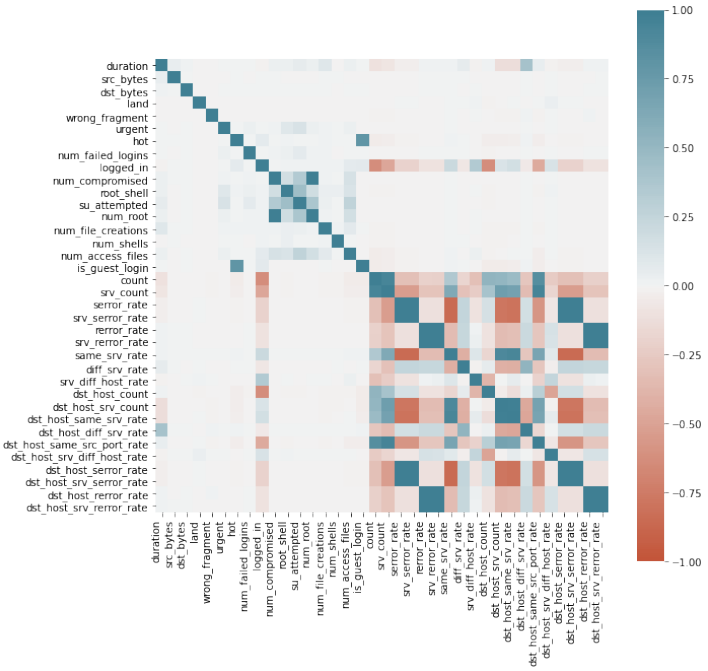


Figure 4: Correlation plot of features in the dataset KDD'99

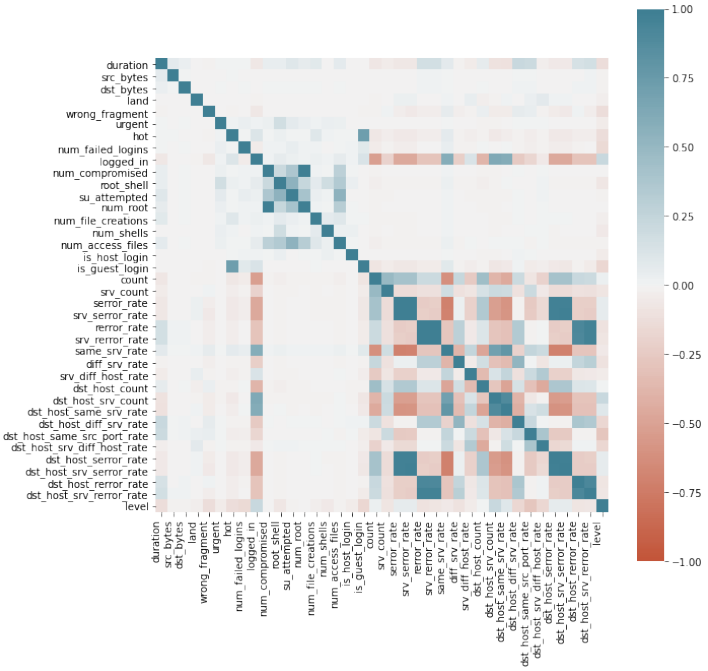


Figure 5: Correlation plot of features in the NSL-KDD dataset

dataset are more correlated as compared to the NSL-KDD dataset, as seen in figures 4 and 5.

It is important to consider that according to [31] machine learning models face a high threat of active adversarial attacks compared to other methods. Thus, further research on ensemble or hybrid methods can help combat the issue mentioned.

7. Conclusion

Compared to the decision tree, MLP, and SAE models, the random forest classifier is a considerably more accurate method for predicting whether a network is undergoing an attack and accurately classifying the attack type. The greater accuracy seen in predictions from the random forest model can be attributed to the fact that they are not affected by dataset imbalance. Random forests can also extract important features to study their relation to the attack and educate those who fall victim to a particular attack type. The top three features extracted from the two datasets by the random forest model were: count, diff_srv_rate, and dst_bytes.

For further study, it is essential to consider the disadvantages of the two datasets by applying these models to more recent open-source datasets. Another direction for future research is using ML and DL models to predict outcomes in real-time.

8. References

[1] Cisco 2018 *Cisco Annual Internet Report (2018–2023).* URL `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`

[2] Shimeall T J and Spring J M 2014 *Introduction to Information Security* 253–274

[3] Stolfo S J 1999 *UCI KDD repository.* URL `http://kdd.ics.uci.edu`

[4] Tavallaee M, Bagheri E, Lu W and Ghorbani A A 2009 *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications* pp 1–6

[5] Mukkamala S, Janoski G and Sung A 2002 *proceedings of the IEEE International Joint Conference on Neural Networks (ANNIE), St. Louis, MO* pp 1702–1707

[6] Chan A, Ng W W, Yeung D S, Tsang E C *et al.* 2005 *Proceedings of 2005 international conference on machine learning and cybernetics* vol 6 pp 18–21

[7] Heba F E, Darwish A, Hassanien A E and Abraham A 2010 *2010 10th international conference on intelligent systems design and applications* (IEEE) pp 363–367

[8] Gao N, Gao L, Gao Q and Wang H 2014 *2014 Second International Conference on Advanced Cloud and Big Data* (IEEE) pp 247–252

[9] Vinayakumar R, Alazab M, Soman K, Poornachandran P, Al-Nemrat A and Venkatraman S 2019 *IEEE Access* **7** 41525–41550

[10] Liu C, Gu Z and Wang J 2021 *IEEE Access* **9** 75729–75740 ISSN 2169-3536

[11] Wang Z, Liu Y, He D and Chan S 2021 *Computers & Security* **103** 102177 ISSN 0167-4048 URL `https://www.sciencedirect.com/science/article/pii/S0167404821000018`

[12] Shone N, Ngoc T N, Phai V D and Shi Q 2018 *IEEE Transactions on Emerging Topics in Computational Intelligence* **2** 41–50

[13] Singh K, Kaur L and Maini R 2021 *Computational Methods and Data Engineering* ed Singh V, Asari V K, Kumar S and Patel R B (Singapore: Springer Singapore) pp 223–241 ISBN 978-981-15-6876-3

[14] Niyaz Q, Sun W and Javaid A Y 2016 *CoRR* **abs/1611.07400** (*Preprint* `1611.07400`) URL `http://arxiv.org/abs/1611.07400`

[15] Hindy H, Atkinson R, Tachtatzis C, Colin J N, Bayne E and Bellekens X 2020 *Electronics* **9** ISSN 2079-9292 URL `https://www.mdpi.com/2079-9292/9/10/1684`

[16] Wu Z, Wang J, Hu L, Zhang Z and Wu H 2020 *Journal of Network and Computer Applications* **164** 102688 ISSN 1084-8045 URL `https://www.sciencedirect.com/science/article/pii/S1084804520301624`

[17] Ahmad I, Basheri M, Iqbal M J and Rahim A 2018 *IEEE access* **6** 33789–33795

[18] Negandhi P, Trivedi Y and Mangrulkar R 2019 *Emerging Research in Computing, Information, Communication and Applications* ed Shetty N R, Patnaik L M, Nagaraj H C, Hamsavath P N and Nalini N (Singapore: Springer Singapore) pp 519–531 ISBN 978-981-13-6001-5

[19] Brugger S T and Chow J 2007 *UCDAVIS department of Computer Science* **1** 22

[20] McHugh J 2000 *ACM Transactions on Information and System Security (TISSEC)* **3** 262–294

[21] Sabhnani M and Serpen G 2004 *Intelligent data analysis* **8** 403–415

[22] Kingma D P and Ba J 2017 Adam: A method for stochastic optimization (*Preprint* `1412.6980`)

[23] Bengio Y, Courville A and Vincent P 2013 *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(8) 1798–1828 ISSN 01628828 dh <= di

[24] Breiman L 2001 *Machine learning* **45** 5–32

[25] Biau G 2012 *The Journal of Machine Learning Research* **13** 1063–1095

[26] Van Rossum G and Drake Jr F L 1995 *Python reference manual* (Centrum voor Wiskunde en Informatica Amsterdam)

[27] Wes McKinney 2010 *Proceedings of the 9th Python in Science Conference* ed Stéfan van der Walt and Jarrod Millman pp 56 – 61 URL 10.25080/Majora-92bf1922-00a

[28] Harris C R, Millman K J, van der Walt S J, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith N J, Kern R, Picus M, Hoyer S, van Kerkwijk M H, Brett M, Haldane A, del Río J F, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C and Oliphant T E 2020 Array programming with NumPy URL https://doi.org/10.1038/s41586-020-2649-2

[29] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado G S, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y and Zheng X 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL https://www.tensorflow.org/

[30] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M and Duchesnay E 2011 *Journal of Machine Learning Research* **12** 2825–2830

[31] Ibitoye O, Abou-Khamis R, Matrawy A and Omair Shafiq M 2019 *arXiv e-prints* arXiv–1911