Intrusion detection using machine learning techniques

Abrar Syed
MS in Computer Science
abrsyed@ucdavis.edu

Mohammed Asim
MS in Computer Science
mdasim@ucdavis.edu

Adesh Thakare
MS in Computer Science
athakare@ucdavis.edu

1 Introduction

In today's digital age, protecting network systems is crucial for data integrity and societal functioning. Intrusion Detection Systems (IDS) play a key role in this protection. IDS employ two main detection types: signature-based, which identifies known threats using established patterns, and anomaly-based, which detects deviations from normal behavior, indicating possible intrusions. Our project investigates enhancing traditional IDS using machine learning (ML). Utilizing the NSL-KDD dataset [1], our project aims to assess the effectiveness of ML in advancing IDS capabilities. However, in light of the known limitations of the NSL-KDD dataset, such as its less-than-ideal representation of real-world networks and bias towards frequent records as per the critique of the 1998 and 1999 DARPA intrusion detection system evaluations performed by MIT's Lincoln laboratory[2]. Our project now extends beyond using NSL-KDD by utilizing the more contemporary and varied CICIDS 2017 dataset [3]. This approach aims to address the deficiencies of NSL-KDD and provides a more comprehensive understanding of network security threats.

1.1 Types of attacks in the NSL-KDD dataset

The NSL-KDD dataset includes attacks such as DoS (Denial-of-Service), U2R (User-to-Root), R2L (Remote-to-Local), and Probing. DoS attacks aim to disrupt the availability of network resources by flooding the network with traffic, overwhelming it and rendering it inaccessible to legitimate users. Probing attacks, on the other hand, are used to gather information about a network's vulnerabilities and configurations, often as a precursor to more targeted attacks. R2L attacks exploit weaknesses in network authentication and access controls to gain unauthorized entry, while U2R attacks leverage system vulnerabilities to gain root-level access, potentially causing significant damage or data breaches.

1.2 Types of attacks in the CICIDS 2017 dataset

The CICIDS 2017 dataset includes the most common attacks based on the 2016 McAfee report, such as Web-based, Brute force, DoS, DDoS, Infiltration, Heart-bleed, Bot, and PortScan. Web-based attacks exploit web application vulnerabilities, and Brute force attacks relentlessly guess login credentials. Infiltration bypasses security to access or damage a network. Heartbleed exploits OpenSSL vulnerabilities to access sensitive data. Bot attacks use automated software for coordinated malicious activities, and PortScan involves scanning for open ports and vulnerabilities. Each attack requires specific detection and prevention strategies in intrusion detection systems.

2 Literature review

There are multiple studies that explore the realm of intrusion detection. Traditional rule-based systems have been the norm [4]. However, there's growing interest in the applicability of ML for IDS and the literature reveals a transformative shift from static rule-based IDS to dynamic ML-driven systems. We found several research papers that cover the broad spectrum of intrusion detection using ML techniques. Tavallaee et al. (2009) introduced the NSL-KDD dataset, which became

a benchmark for evaluating machine learning models in intrusion detection. Wagh et al. (2013) [5] made a comprehensive comparison between rule-based systems and ML models. Ayesha et al. (2021) [6] presented a comprehensive critical survey of ML-based intrusion detection approaches in the literature in the last ten years. We went through all these papers to get a better and deeper understanding of our topic.

3 Datasets used

3.1 The NSL-KDD dataset

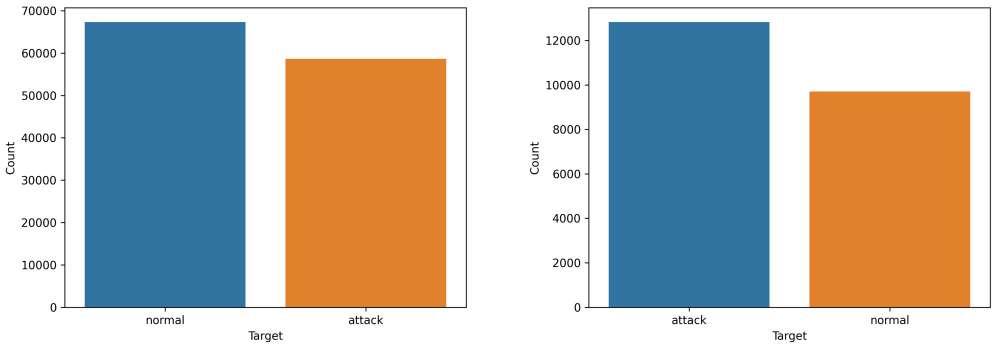
We utilized the NSL-KDD dataset, an updated version of the KDD'99 dataset, obtained from the official NSL-KDD website. This dataset offers several enhancements over the KDD'99 dataset such as:

- The NSL-KDD dataset does not contain any null values within both the train and test sets.
- Redundant records have been eliminated from both the train and test sets, mitigating bias toward more frequent records.

The training set comprises 125,973 entries, while the test set consists of 22,543 entries. Each set encompasses 43 columns, namely: duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, outcome, level.

The NSL-KDD dataset includes a set of network connections, each labeled as either normal or as an attack, with one of four categories: Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L).

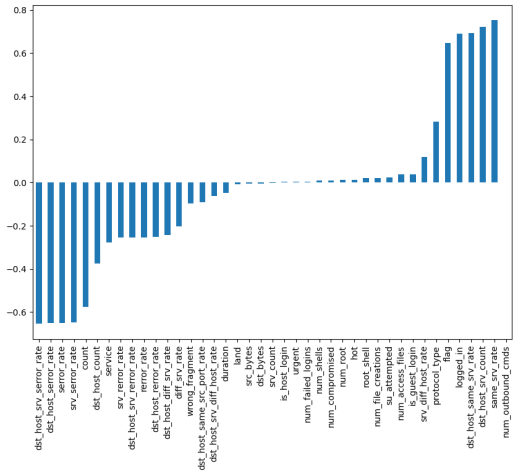
Fig 1 and Fig 2 show the distribution of the NSL KDD dataset and the correlation between features respectively.



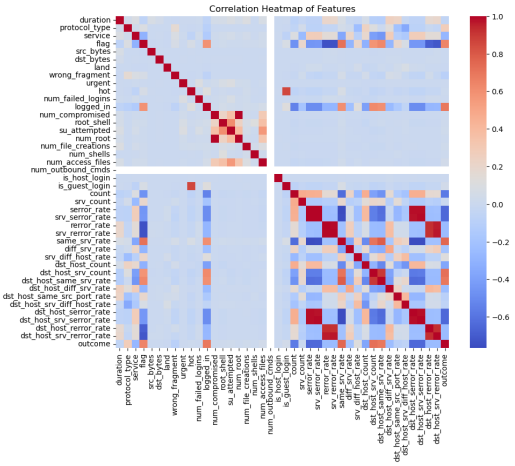
(a) Distribution of the training data

(b) Distribution of the test data

Figure 1: Distribution of training and test data NSL KDD



(a) Correlation of features with the target variable for NSL-KDD



(b) Correlation heatmap for NSL-KDD

Figure 2: Correlation plots for NSL-KDD

3.2 The CICIDS dataset

Apart from the NSL-KDD dataset, we also applied two of our four machine learning models to the CICIDS 2017 dataset. Since we had limited computational power we were not able train on the entire dataset. Thus we selected a subset of the CICIDS 2017 dataset. CICIDS 2017 dataset contains benign (normal) values and the most up-to-date common attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack (CSV files).

The data capturing period started at 9 a.m., Monday, July 3, 2017 and ended at 5 p.m. on Friday July 7, 2017, for a total of 5 days. Monday is the normal day and only includes the benign traffic. The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS. They have been executed both morning and afternoon on Tuesday, Wednesday, Thursday and Friday.

We did our analysis only on the Friday dataset files. The dataset consists of 79 columns, including the target column labeled 'Label.' We divided the dataset into a 70-30 split, creating a training set that comprises 70% of the data for training purposes and a separate 30% portion designated for the test set. Out of the 79 features, we selected the top 22 most relevant features using a random forest classifier.

Figure 4 shows the distribution of the training data and the test data for the CICIDS dataset.

4 Dataset preparation and preprocessing

In order to prepare the data for the machine learning algorithms, a set of operations were carried out:

1. Elimination of variables and feature selection:
   Since our project is a classification problem, we decided to drop the 'level' column from the NSL-KDD dataset which specifies the level of the threat. Our goal is to classify an intrusion as normal or attack only. We are not dealing with the regression problem here. The column 'num_outbound_cmds' was a constant column with each value being 0, therefore we decided to drop this column as it would not aid in making predictions using the different models.
   Then, we applied a random forest classifier to get the feature importance of each column and filtered the dataset using a threshold value of 0.005. After this step, our dataset had 25 features remaining. Similary, a random forest classifier was applied to the the CICIDS

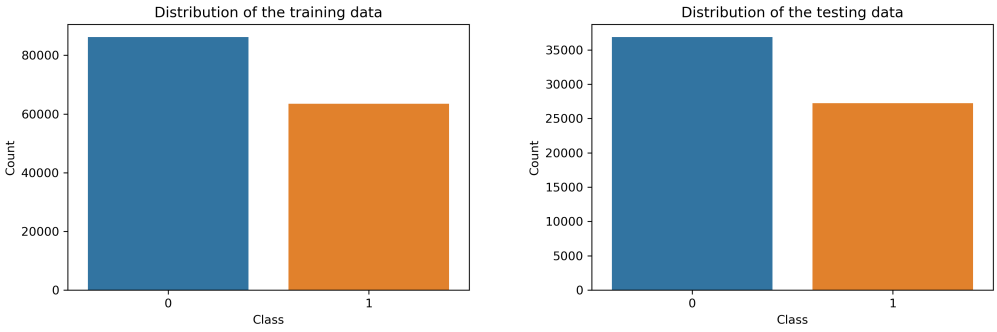


Figure 3: Distribution of training and test data CICIDS

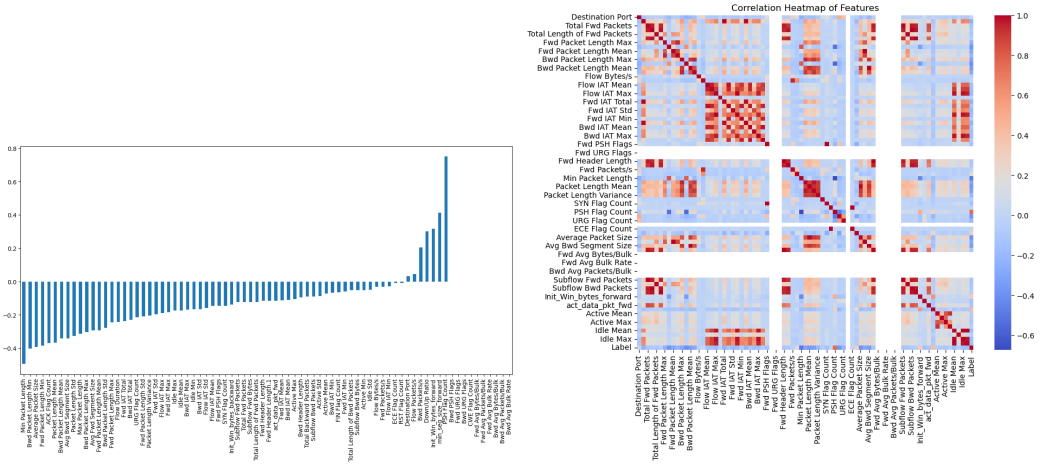


Figure 4: Distribution of training and test data CICIDS

dataset to get the most important features based on a threshold value of 0.015. After this step, the dataset had 22 features remaining.

2. Conversion of categorical variables to numeric
   It was necessary to carry out this operation for the data to be acceptable by our machine learning algorithms. The categorical variables in the NSL KDD dataset are protocol_type, service, flag and our target variable i.e, outcome. All these variables were converted to numerical values using the LabelEncoder present in the sklearn library [7].

   For the CICIDS, the only categorical variable was the target column 'Label' which was converted to numerical values where 0 maps to benign and 1 is mapped to attacks.

3. Data normalization
   A normalization of the data was carried out using the RobustScaler present in sklearn's[7] preprocessing library. This Scaler removes the median and scales the data according to the quantile range. The Interquatile Range (IQR) is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set.

5 Evaluation metrics used

5.1 Accuracy

Accuracy measures the proportion of correctly classified instances out of the total instances in a dataset, providing an overall performance evaluation of a classification model. Accuracy is a useful metric when the dalanced in question is a balanced dataset. For an unbalanced dataset metrics like Precision, Recall and F1-score are used. Accuracy is calculated as follows:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FN+FP}$$

where:
TP: True Positive. Result in which the model correctly predicts the positive class.
FP: False positive. Result in which the model incorrectly predicts the positive class.
TN: True Negative. Result in which the model correctly predicts the negative class
FN: False Negative. Result in which the model incorrectly predicts the negative class.

5.2 Precision

Precision represents the ratio of correctly predicted positive observations to the total predicted positive observations, highlighting the model's capability to avoid false positives. It is calculated as follows:

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

5.3 Recall

Recall, also known as sensitivity or true positive rate, measures the ratio of correctly predicted positive instances to the actual positives in the dataset, emphasizing the model's ability to capture all positive instances. It is calculated as follows:

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

5.4 F1 score

F1 score is the harmonic mean of precision and recall, providing a balanced measure that considers both precision and recall, making it a useful metric for assessing a model's performance when the class distribution is imbalanced. It is calculated as follows:

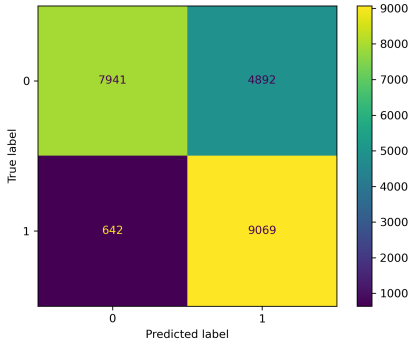
$$\text{F1-Score} = \frac{2*precision*recall}{precision+recall}$$

6 Results

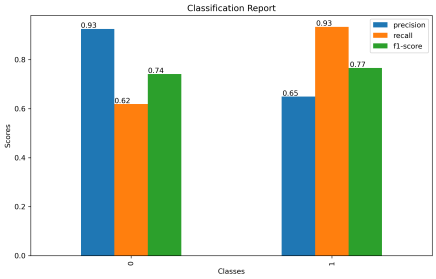
6.1 Results obtained on the NSL-KDD dataset

6.1.1 Logistic Regression

Logistic regression was implemented using the Python LogisticRegression function defined in the scikitlearn [7] library. Logistic regression is our baseline model for both the datasets, we would be comparing all other models with respect to logistic regression. The confusion matrix obtained using logistic regression is shown in Figure 5a.
Confusion matrix is a matrix that allows us to visualize the performance of a model by comparing its predictions against the actual values. The classification report is shown in Figure 5b.



(a) Confusion matrix for logistic regression



(b) Classification report for logistic regression

Figure 5: Confusion matrix and classification report for logistic regression

6.1.2 K Nearest Neighbors (KNN)

The K Nearest Neighbors (KNN) algorithm was implemented using the Python knn function defined in the scikitlearn [7] library. To determine the optimal value of k, we applied the elbow method. Figure 6 shows the elbow method applied on KNN. The accuracy score obtained for knn is 0.77. The confusion matrix obtained using knn is shown in Figure 7a. The classification report is shown in Figure 7b.

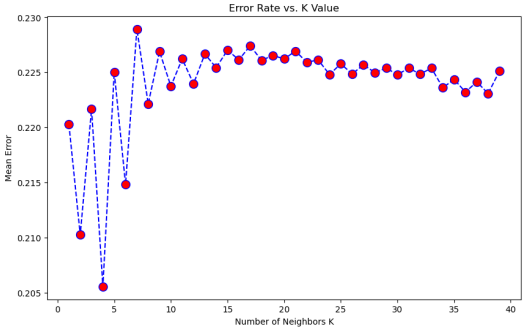
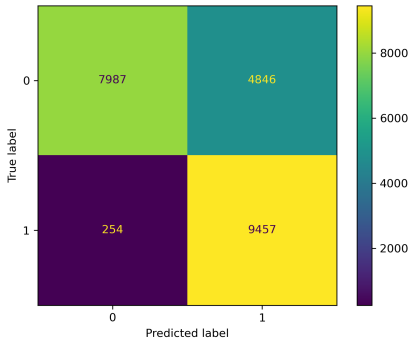
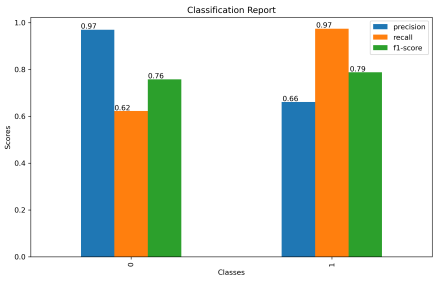


Figure 6: Error vs K value



(a) Confusion matrix for KNN

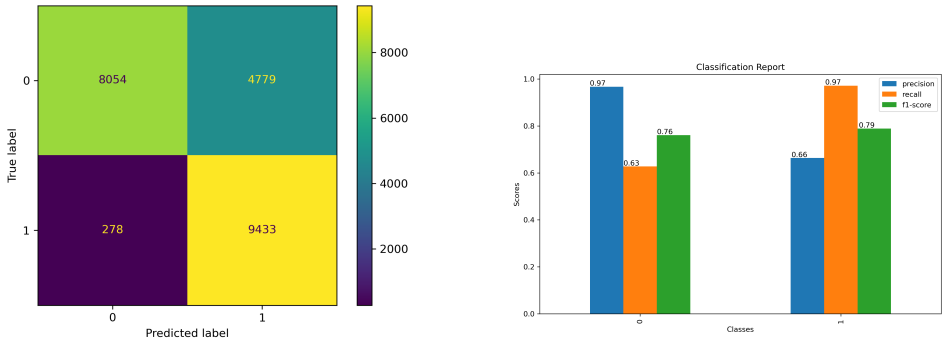


(b) Classification report for KNN

Figure 7: Confusion matrix and classification report for KNN

6.1.3 Random forest classifier

The random forest algorithm was implemented using the RandomForestClassifier function defined in scikit-learn[7] library. The accuracy score obtained for random forest is 0.78. The confusion matrix obtained using random forest is shown in Figure 8a. The classification report is shown in Figure 8b.



(a) Confusion matrix for random forest

(b) Classification report for random forest

Figure 8: Confusion matrix and classification report for random forest

6.1.4 Artificial neural network(ANN)

This algorithm was implemented using tensorflow[8] in python. Since there are hyperparameters such as number of hidden layers, number of hidden neurons in each layer, learning rate etc. to tune in an ANN, we utilized the Keras tuner[9] library in Keras to obtain the best set of hyperparameters of our model.

Keras tuner does a random search in the search space that we initially set. We had set the search space as following:

- For the number of hidden layers, we search for the best values between the range 2 to 10.
- For the number of hidden neurons in each hidden layer, we initialized the range to search between 2 to 20.
- For the learning rate, the tuner searched for the optimal value between the values 0.01, 0.001, 0.0001.

The best set of hyperparameters obtained using Keras tuner are as follows:

- number of layers: 4
- neurons in hidden layer 1: 16
- neurons in hidden layer 2: 18
- neurons in hidden layer 1: 18
- neurons in hidden layer 1: 8
- neurons in hidden layer 1: 16
- learning rate: 0.0001

All layers except the output layer had the relu activation function[10] that returns the input value if it is positive or zero, and returns zero for any negative input. Mathematically, it is defined as

$$ReLU(x) = max(0, x).$$

The output layer had sigmoid activation function that takes any input value and maps it to a value between 0 and 1. Mathematically, it is defined as

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

The loss function used was the binary crossentropy, which measures the difference between predicted probabilities and the true binary labels of a classification model. It is defined as:

$$L(y, \hat{y}) = -(y * log(\hat{y}) + (1 - y) * log(1 - \hat{y}))$$

The ANN was trained for 100 epochs using the Keras early stopping which prevents overfitiing of the model on the training data. The loss curve against the number of epochs is shown in Figure 9. The training stopped at around 10 epochs after which there was no further reduction in the validation loss. It can be seen from Figure 9 that there is neither underfitting nor overfitting during the training of the model. The confusion matrix and the classification report for ANN is shown in Figure 10a and Figure 10b respectively.

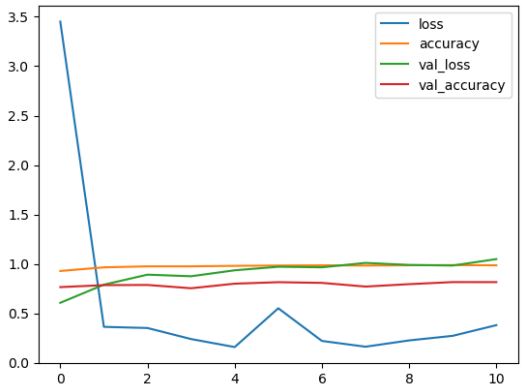
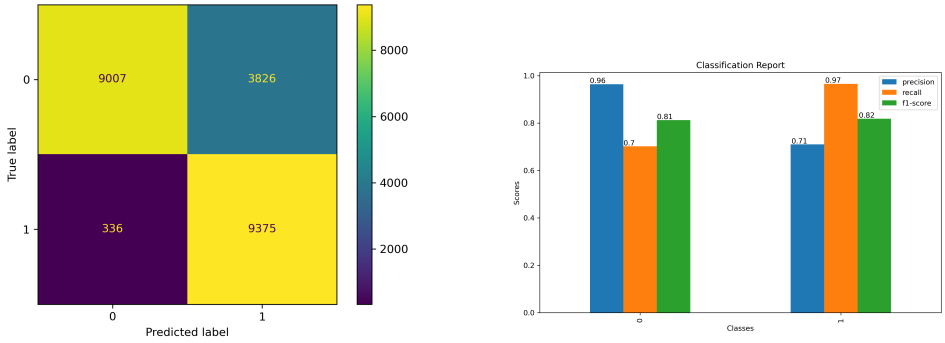


Figure 9: Loss curve against number of epochs



(a) Confusion matrix for ANN

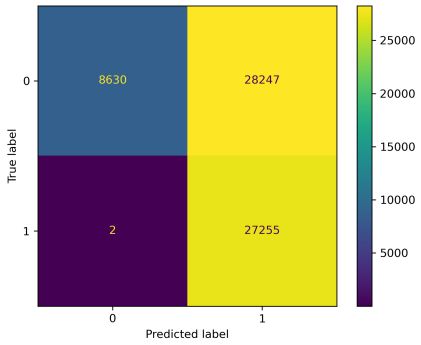
(b) Classification report for ANN

Figure 10: Confusion matrix and classification report for ANN

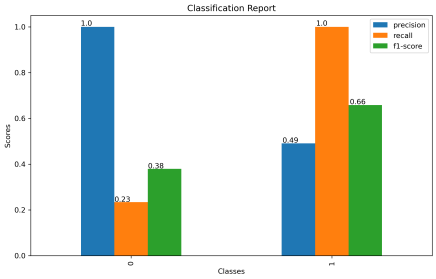
6.2 Results obtained on the CICIDS 2017 dataset

6.2.1 Logistic regression

The confusion matrix obtained for logistic regression is shown in Figure 11a. The classification report is shown in Figure 11b.



(a) Confusion matrix for Logistic regression, CICIDS dataset

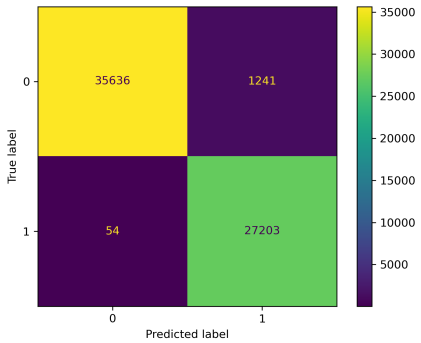


(b) Classification report for Logistic regression, CICIDS dataset

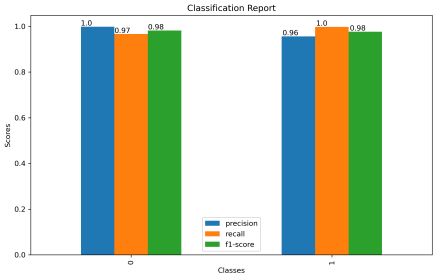
Figure 11: Confusion matrix and classification report for logistic regression on CICIDS dataset

6.2.2 Artificial neural network

An ANN was developed having 4 hidden layers and 50, 25, 10 and 5 neurons in each hidden layer respectively. The confusion matrix obtained for ANN is shown in Figure 12a. The classification report is shown in Figure 12b.



(a) Confusion matrix for ANN, CICIDS dataset



(b) Classification report for ANN, CICIDS dataset

Figure 12: Confusion matrix and classification report for ANN on CICIDS dataset

Table 1 shows the values of performance metrics for different machine learning algorithms applied in the present project.

Algorithm	Accuracy	Precision	Recall	F1-Score
Logistic Regression (NSL-KDD)	75	65	93	77
K nearest neighbors	77	66	98	79
Random Forest	78	66	97	79
ANN (NSL-KDD)	82	71	97	82
Logistic Regression (CICIDS)	56	49	1	66
ANN (CICIDS)	98	96	1	98

Table 1: Comparison of Model Performance

It can be seen from Table 1 that for both the datasets our ANN model performed best because ANNs are powerful models known for their ability to capture complex nonlinear patterns in data.

References

[1] Mahbod Tavallaee et al. "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.

[2] John McHugh. "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory". In: *ACM Trans. Inf. Syst. Secur.* 3.4 (Nov. 2000), pp. 262–294. ISSN: 1094-9224. DOI: 10.1145/382912.382923. URL: https://doi.org/10.1145/382912.382923.

[3] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *International Conference on Information Systems Security and Privacy*. 2018. URL: https://api.semanticscholar.org/CorpusID:4707749.

[4] Ansam Khraisat et al. "Survey of intrusion detection systems: techniques, datasets and challenges". In: *Cybersecurity* 2.1 (2019), pp. 1–22. DOI: https://doi.org/10.1186/s42400-019-0038-7.

[5] Sharmila Kishor Wagh, Vinod K Pachghare, and Satish R Kolhe. "Survey on intrusion detection system using machine learning techniques". In: *International Journal of Computer Applications* 78.16 (2013), pp. 30–37. DOI: 10.5120/13608-1412.

[6] Ayesha S Dina and D Manivannan. "Intrusion detection based on machine learning techniques in computer networks". In: *Internet of Things* 16 (2021), p. 100462. DOI: https://doi.org/10.1016/j.iot.2021.100462.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[9] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[10] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

[11] Robert A Bridges et al. "A survey of intrusion detection systems leveraging host data". In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–35. DOI: https://doi.org/10.48550/arXiv.1805.06070.

[12] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998. URL: https://dl.acm.org/doi/10.5555/521706.

[13] Yoney Kirsal Ever, Boran Sekeroglu, and Kamil Dimililer. "Classification analysis of intrusion detection on NSL-KDD using machine learning algorithms". In: *International Conference on Mobile Web and Intelligent Information Systems*. Springer. 2019, pp. 111–122. DOI: 10.1007/978-3-030-27192-3_9.

[14] Razan Abdulhammed et al. "Deep and machine learning approaches for anomaly-based intrusion detection of imbalanced network traffic". In: *IEEE sensors letters* 3.1 (2018), pp. 1–4. DOI: https://doi.org/10.3390/electronics8030322.