Comparative Analysis of Classical Machine Learning Algorithms and Artificial Neural Networks for Binary Intrusion Detection Using the NSL-KDD Dataset

Marcel Ewinge
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Table of Contents

1. Motivation

In our highly digital world, cyberattacks are becoming more frequent. This creates serious risks to the security and reliability of today's information systems. Therefore, it is important to develop accurate intrusion detection systems (IDS) to protect digital assets. [1]. In this work, I focus on a core challenge: using data-driven methods to classify network traffic as either "Normal" or "Attack" based on the widely-used NSL-KDD benchmark dataset.

A review of existing research shows there's no clear agreement on the best algorithm for this binary classification task. Traditional machine learning models, such as Random Forests and Support Vector Machines, have achieved excellent results, with some studies reporting accuracies up to 99.7% [2]. However, their dominance isn't guaranteed. Well-designed Artificial Neural Networks can reach similar performance levels [1], [3], though they are highly sensitive to architecture and hyperparameter choices.

Overall, the evidence suggests that the key to strong results lies less in the type of algorithm and more in the details of how experiments are designed and conducted, a factor that explains much of the variability seen in the field.

My study i perform a comparative analysis of classical machine learning algorithms and an Artificial Neural Network (ANN) for binary intrusion detection utilizing the NSL-KDD dataset. The classical models were good starting points, but Weights & Biases were used to systematically optimize the ANN to find its best configuration. The test showed that the Decision Tree had the highest F1-score (0.8071), while the optimized ANN did almost as well (0.7937).

2. Data

The NSL-KDD dataset, also well-known as a benchmark in the field of the network intrusion detection systems (NIDS) research, is used in this project [4]. To address some of the inherent statistical problems of the original KDD'99 dataset, the University of New Brunswick developed it as an improved version [5]. In order to prevent machine learning algorithms from being biased towards the more frequent records and from learning rare but significant attack patterns. The new dataset was primarily created to eliminate the enormous number of redundant and duplicate records that were present in the original KDD'99 dataset [2].

While various versions of the NSL-KDD dataset are available on platforms like Kaggle, differing in format and scope, this project utilizes the core **KDDTrain+.txt** and **KDDTest+.txt** files, which are also used in most other projects. A key advantage of using these specific files is that the dataset is already separated into training and testing sets [6]. The dataset contains network connection records, each described by 41 features and labeled with a specific attack type or as 'normal'. These features are a mix of continuous, discrete, and symbolic data types. They capture a broad range of properties from basic connection duration and protocol type (e.g., TCP, UDP, ICMP) to higher-level traffic characteristics like the rate of connection errors. For this project, the primary goal is binary classification between normal and attack. Therefore, the original multi-class labels, which include categories such as 'Normal', 'DoS', 'Probe', 'R2L', and 'U2R' [1], were transformed into a single binary target variable. Connections labeled 'normal' were mapped to 0, and all attacks connections were combined to 1. This resulted in a reasonably balanced class distribution in the training set, with approximately 53% normal traffic and 47% attack traffic.

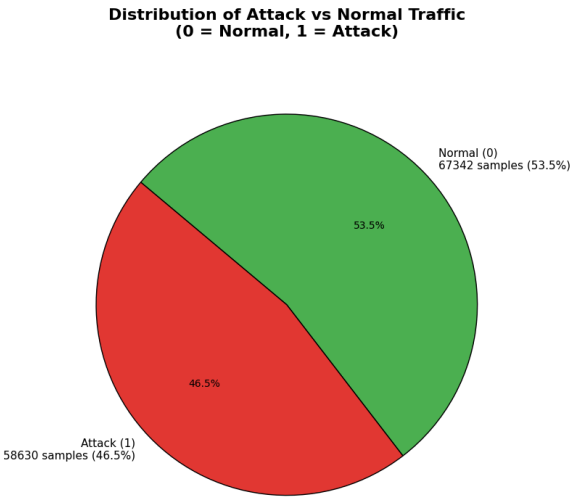

Figure 1: Distribution of Attack vs Normal Traffic

A exploratory analysis revealed several key characteristics of the data that necessitated a structured preprocessing pipeline. The dataset was found to be **complete**, with **no missing** or **null values**. However, an analysis of the numerical features showed big differences in scale. For instance, features like **src_bytes** can have values in the millions, while many rate-based features are normalized between 0 and 1.

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in
count	125972.000000	1.259720e+05	1.259720e+05	125972.000000	125972.000000	125972.000000	125972.000000	125972.000000	125972.000000
mean	287.146929	4.556710e+04	1.977927e+04	0.000198	0.022688	0.000111	0.204411	0.001222	0.395739
std	2604.525522	5.870354e+06	4.021285e+06	0.014086	0.253531	0.014366	2.149977	0.045239	0.489011
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	2.760000e+02	5.160000e+02	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	42908.000000	1.379964e+09	1.309937e+09	1.000000	3.000000	3.000000	77.000000	5.000000	1.000000

Figure 2: Statistical overview of numerical network features

This disparity makes feature scaling a mandatory step to prevent algorithms that are sensitive to feature magnitude. Consequently, the **StandardScaler** was selected to transform all numerical features to have a mean of zero and a standard deviation of one.

```
Statistical key figures of the numerical features BEFORE scaling
        duration      src_bytes      dst_bytes      land  wrong_fragment  \
mean    287.146929  4.556710e+04  1.977927e+04  0.000198        0.022688
std    2604.525522  5.870354e+06  4.021285e+06  0.014086        0.253531
Statistical key figures of the numerical features AFTER scaling (by preprocessor)
          duration      src_bytes      dst_bytes          land  wrong_fragment  \
mean  4.083709e-17 -9.024770e-19 -1.692144e-18  3.384289e-18    2.171585e-18
std   1.000004e+00  1.000004e+00  1.000004e+00  1.000004e+00    1.000004e+00
```

Figure 3: Mean and Std. before and after the use of StandardScaler

Furthermore, the feature **num_outbound_cmds**, was identified as having zero variance across all records, providing no predictive information and was therefore removed from both the training and testing sets. The three categorical **features—protocol_type, service, and flag—were** were transformed into a numerical format using the a technic called **one-hot encoding**. This is very important for algorithms that nned numerical valuesfor their input. A negativ effect of this technic is the increase of the dimensionality of the dataset. The number of features increaed from 40 to 121. This is primarily driven by the high cardinality of the service type feature, which contains 70 unique categories.

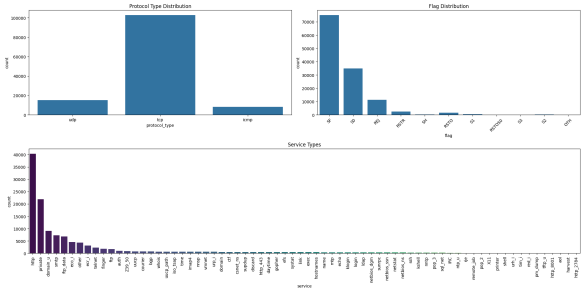

Figure 4: Categorial Features Analysis

3. Theoretical Part

A analysis of different machine learning algorithms for binary intrusion detection on the NSL-KDD dataset reveals a complex and varied performance landscape, where no single algorithmic class demonstrates absolute superiority. As stated in Chapter 2, the dataset eliminated unnecessary records that might bias classifiers. The NSL-KDD dataset was created to address the shortcomings of the previous KDD'99 dataset [5], [7]. In this review classical machine learning models, including Decision Trees, K-Nearest Neighbors, Support Vector Machines, Random Forests, and Logistic Regression,are tested against a basic Artificial Neural Network for the binary task of differentiating "Normal" network traffic from "Attack" traffic. While the results depend a lot on things like the experimental setup, feature selection, and hyperparameter

tuning, the evidence suggests that a well-configured artificial neural network can often match, or even beat, classical algorithms.

Intrusion detection systems typically rely on either anomaly detection or signature matching to differentiate between normal or malicious network activity. The NSL-KDD benchmark dataset is widely used in research because it removes the significant redundancy and some distribution shifts found in the original KDD'99 dataset, while still preserving the same 41 features and attack taxonomy [7]. Even if the dataset has some advantages over its predecessor, preprocessing remains important. The official binary classification training set contains 67.343 normal and 58.630 intrusion records, resulting in a more balanced class ratio compared to the multi-class version. Before training, most studies apply one-hot encoding to categorical features and normalize numerical variables using either L2 normalization or z-score standardization [5], [2], [4].

The algorithms reviewed in this work adopt a different approach to traffic classification. Decision Trees and Random Forests, build rule-based models to distinguish between classes. By aggregating the predictions of multiple trees, RFs generally achieve higher accuracy and greater robustness against overfitting [8]. Support Vector Machines (SVM), a potentially effective but computationally demanding technique, seek to identify the best separating hyperplane between classes in a high-dimensional feature space [5]. On the other hand, complex, non-linear relationships in the data are learned by Artificial Neural Networks (ANN), which are made up of interconnected layers of nodes. Their ability to model extremely complex patterns is highly dependent on their architecture, including the number of layers and neurons as well as the optimization process [1], [3].

Significant variation is revealed by a direct comparison of binary classification performance across the literature that is provided. According to certain research, classical machine algorithms perform better. For example, Random Forest has an impressive 99.7% accuracy on NSL-KDD, according to a review by [2]. Likewise, they discovered that an SVM outperformed their ANN (0.7557) and Random Forest (0.7528) implementations, achieving the highest F1-score (0.8476) in their comparison. Although they also warn that their SVM and RF models were trained on a small data sample due to computational expense (SVM at 0.05%) and random forest at (0.1%) of the training set, they specifically mentioned that their ANN and RF models suffered from low recall, indicating a difficulty in correctly identifying attack instances) [5].

However, other studies show that ANNs have a lot of potenntial. [4] set a baseline for ANN performance, reducing the feature set to 29 attributes and achieving an accuracy of 81.2% for binary classification. The findings of [7] show similar weighted F1-scores of about 81–82% for ANN, SVM, and Random Forest on the NSL-KDD binary task. These results temper claims of superiority and suggest performance parity under the specific experimental conditions.

The absence of standardized evaluation procedures is primarily to blame for these disparities in reported results. Studies differ in the number of input features, ranging from 20 [7] to 38 [1]. The comparability of results is also impacted by the different data splitting techniques used by different researchers. For example, some use the standard KDDTrain+/ KDDTest+ files [5], while others use custom 70/30 splits [1].

In conclusion, there is some support but not enough evidence to conclusively prove that classical machine learning algorithms can perform on par with or better than a basic ANN for binary intrusion detection on the NSL-KDD dataset. Based on different papers, classical machine learning models such as Random Forest and SVM can perform better than some ANN. This can happen, when the

ANN is not set up optimally or when evaluation metrics like F1-score are given priority [2], [5].

In the end the performance depends not so much on the type of algorithm you choose, but more on how you implement it. Topics like feature engineering, data handling and the careful tuning of hyperparameters make the biggest difference. While classical models are often more robust and less complex, ANNs have the potential for higher performance. This comes at the cost of increased complexity and tuning effort.

4. Implementation

In my implementation I used python as the programming language along with the following libraries:

- TensorFlow/Keras for building and training the neural network
- Scikit-learn for data preprocessing, evaluation and for classical machine learning algorithms
- Pandas and NumPy for data manipulation and analysis
- Weights & Biases (W&B) for experiment tracking and hyperparameter optimization

The core for the classical machine learning algorithms implementation relies on Scikit-learn. It provides the modules for data preprocessing (**StandardScaler, OneHotEncoder**), building composite pipelines (Pipeline, ColumnTransformer), training the classical models (LogisticRegression, SVC, DecisionTreeClassifier, etc.) and calculating performance metrics. For visualizing the libary Matplotlib and Seaborn was used.

A systematic approach was used to train and test the classical machine learning models, such as Logistic Regression, Support Vector Machine (SVM), Decision Tree, Random Forest, and K-Nearest Neighbors (KNN). Each model was put into a Scikit-learn Pipeline that linked the preprocessor to the model's classifier.

The implementation of the Artificial Neural Network (ANN) was handled differently. It did not use Scikit-learn, but the TensorFlow and Keras libraries. The overall architecture, especially the number of hidden layers, was directly adapted from the literature. Most of the papers used two hidden layers for binary classification. The performance of ANNs is highly sensitive to their architecture and the choice of hyperparameters. To address this, the Weights & Biases (W&B) platform was used to search for the optimal model configuration. The goal of the optimization process, also known as a "Sweep", was to maximize the validation accuracy (val_accuracy) on the held-out test set. The hyperparameter were parameters such as the number of epochs, batch_size, learning_rate, choice of optimizer, the number of neurons in each of the two hidden layers (hidden_layer_1_size, hidden_layer_2_size), the activation function, and the dropout_rate for regularization.

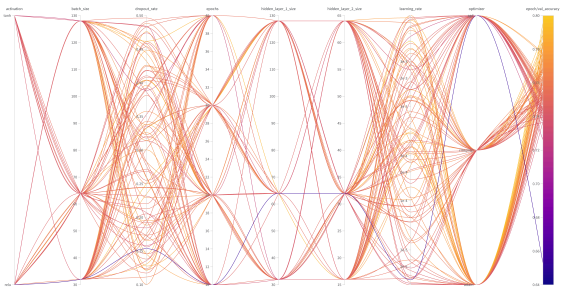

Figure 5: Finished Sweep of the ANN

A total of 100 individual training runs were executed by the W&B agent. The goal was to to explore the defined hyperparameter space. The results of this search are visualized in Figure 5, which illustrates the relationship between different hyperparameter values and also the resulting model performance. In general the W&B sweep identifies the optimal hyperparameter to maximaze the final validation accuracy (val_accuracy), which is achieved at the end of training. When i retrained the model, i found it interestingly, that the re-trained model, which was using the supposedly optimal configuratio, had worrse reults than my inferior model with the initial default hyperparameters. After that i

observed the validation accuracy curve for all 100 runs, whihc revealed a crucial insight. One run, which had not the highest final accuracy, achieved the overall peak accuracy of 0.81 at epoch 18 out of 30.

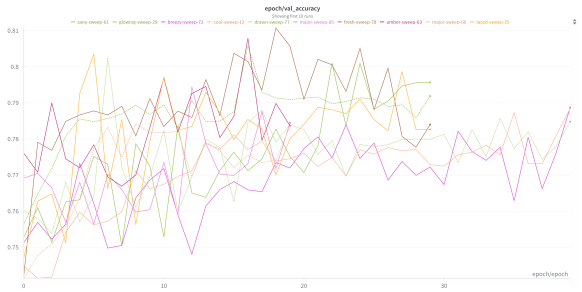

Figure 6: Top 10 runs epoch/val_accuracy

After this finding, i retrained the ANN with the new optimal configuration. The new model slightly outperformed what was previously considered the best configuration. The best-performing model based on Figure 6 utilized the relu activation function, a batch size of 32, a dropout rate of approximately 0.33261386622820965, and was trained for 18 epochs. The optimization was most effective with the adam optimizer using a learning rate of about 0.004939561794461328, and the optimal architecture consisted of two hidden layers, each with 32 neurons.

To provide a more detailed view beyond the summary metrics, a granular analysis of the optimized ANN is presented.

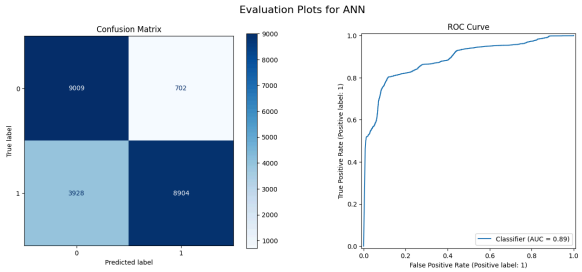

Figure 7: Confusion Matrix of the ANN

Based on the Confusion matrix the model correctly identified 9.009 normal connections (True Negatives) and 8.904 attack connections (True Positives). The model incorrectly identified 702 normal connections as attacks (False Positives), which would cause false alarms in the real world. More importantly, it didn't catch 3.928 real attacks and instead thought they were normal (False Negatives). This shows the natural trade-off between recall and precision. The ROC curve (AUC = 0.89) shows that the model can tell the difference between good and bad traffic very well.

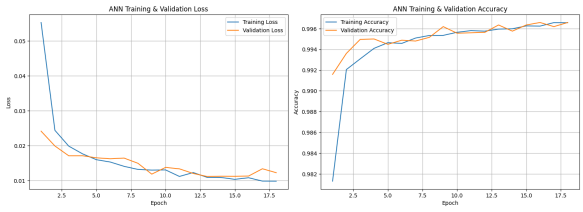

Figure 8: Accuracy and Loss Model

As seen in the figure, the model is successfully learning and minimizing its error, as seen by the "Loss" plot on the left, where the validation loss (orange) and training loss (blue) both steadily decline and converge. Importantly, the validation loss does not begin to rise, indicating that there was no substantial **overfitting** in the model. This trend is reflected in the "Accuracy" plot on the right side, which shows that both training and validation accuracy are increasing quickly and leveling off at a high level. The validity of the final evaluation metrics is supported by the training and validation curves' close proximity in both plots, which shows how well the model generalizes from the training data to the test data.

5. Evaluation

In the evaluation phase, I assessed the performance and outcomes of my implemented approach for intrusion detection using the NSL-KDD dataset. I analysed and visualized the results to understand the effectiveness of the approach compared to baselines from the literature. The primary goal is to determine which model is most effective for the task of binary intrusion detection on the NSL-KDD dataset. To ensure a comprehensive assessment, a standard set of classification metrics was employed like, Accuracy, Precision, Recall, F1-Score and AUC.

All models were trained on the full preprocessed training set and subsequently eval-

uated on the unseen, preprocessed test set. The results of this evaluation are summarized in Figure. This table presents the performance of the models developed in this project alongside.

```
================================================
      Final Comparative Summary of All Models
================================================
                     accuracy  precision  recall  f1_score     auc
Decision Tree          0.8115     0.9665  0.6929    0.8071  0.8308
ANN                    0.7946     0.9269  0.6939    0.7937  0.8875
KNN                    0.7802     0.9679  0.6350    0.7669  0.8274
SVM                    0.7721     0.9629  0.6236    0.7570  0.8886
Random Forest          0.7650     0.9675  0.6076    0.7464  0.9593
Logistic Regression    0.7536     0.9174  0.6232    0.7422  0.7915
```

Figure 9: Summary Table of all algorithms

The baseline results are taken from a study by [5], which conducted a similar binary classification task on the NSL-KDD dataset.

Model	Precision	Recall	F1-Score
SVM	0.8839	0.8142	0.8476
Random Forest	0.9683	0.6158	0.7528
ANN	0.9661	0.6205	0.7557

Based on F1-Score the Decision Tree emerges as the top-performing (0.8071) closely followed by the optimized ANN (0.7557).

One of the most interesting findings was the notable trade-off between precision and recall. It was impressive how precise most of our models were, especially the Decision Tree and KNN. This is crucial, because it means when they classify something as an attack, they're very likely correct. It helps to keep those false alarms to a minimum. However, that excellent precision came at a real cost: significantly lower recall, mostly hovering between 0.60 and 0.69. What this tells us is critical: while these models are precise in what they do detect, they're simply failing to catch a significant portion of the actual attacks happening. For a system designed to detect real-world intrusions, that's a monumental weakness.

When compared to the baseline from [5], the results are more nuanced. The SVM from the baseline study, with an F1-Score of 0.8476, shows a better balance between precision and recall than any model from my project. Nonetheless, the optimized ANN and Decision Tree from this study still outperform the baseline's ANN and Random Forest models in terms of F1-Score. This highlights that while my models are highly precise, further work would be needed to improve their ability to detect all instances of malicious activity.

6. Conclusion

This project successfully implemented and evaluated a range of classical machine learning algorithms and an optimized Artificial Neural Network for binary intrusion detection on the NSL-KDD dataset. In Part 5 it was revealed that the Decision Tree is the top-performing model based on the F1-score, closely followed by ANN. The big takeaway from these results is a classic precision-recall dilemma. The models were really good at being precise, which is fantastic for an intrusion detection system, because the system would not give many false alarms. But unfortunately, their recall was only moderate, meaning they were still letting a lot of real attacks go undetected.

For future work, a focus on prioritizing improving the recall should be set. Improving that will make detection system much more balanced and genuinely useful. One option would be maybe using data-level techniques like SMOTE to balance outline subtle class differences, or perhaps better feature engineering. Algorithmic adjustments, like class weighting, are also an option. Beyond that, exploring into more advanced ensemble models, like XGBoost, is also possible. In the end, moving from binary to multi-class classification would be a big step, building a truly comprehensive intrusion detection system similar to those highlighted in the literature.

List of Figures

Bibliography

[1] A. J. Mohammed, M. H. Arif, and A. A. Ali, "A multilayer perceptron artificial neural network approach for improving the accuracy of intrusion detection systems," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 4, p. 609, 2020.

[2] R. D. Ravipati and M. Abualkibash, "Intrusion detection system classification using different machine learning algorithms on KDD-99 and NSL-KDD datasets-a review paper," *International Journal of Computer Science & Information Technology (IJCSIT) Vol*, vol. 11, 2019.

[3] M. Choraś and M. Pawlicki, "Intrusion detection approach based on optimised artificial neural network," *Neurocomputing*, vol. 452, pp. 705–715, 2021.

[4] B. Ingre and A. Yadav, "Performance analysis of NSL-KDD dataset using ANN," in *2015 international conference on signal processing and communication engineering systems*, 2015, pp. 92–96.

[5] S. Sapre, P. Ahmadi, and K. Islam, "A robust comparison of the KDDCup99 and NSL-KDD IoT network intrusion detection datasets through various machine learning algorithms," *arXiv preprint arXiv:1912.13204*, 2019.

[6] M. H. Zaib, "Kaggle." [Online]. Available: https://www.kaggle.com/datasets/hassan06/nslkdd/data

[7] A. Divekar, M. Parekh, V. Savla, R. Mishra, and M. Shirole, "Benchmarking datasets for anomaly-based network intrusion detection: KDD CUP 99 alternatives," in *2018 IEEE 3rd international conference on computing, communication and security (ICCCS)*, 2018, pp. 1–8.

[8] V. Pai, N. Adesh, and others, "Comparative analysis of machine learning algorithms for intrusion detection," in *IOP Conference Series: Materials Science and Engineering*, 2021, p. 12038.