NetSafe: Network Intrusion Classification

Angadjeet Singh angadjeet22071@iiitd.ac.in
Apaar Garg apaar22089@iiitd.ac.in
Arnav Shukla arnav22103@iiitd.ac.in
Parth Sandeep Rastogi parth22352@iiitd.ac.in

Abstract

The increasing variety of network attacks requires developing more robust intrusion detection systems. This project aims to improve detection accuracy in multi-class classification scenarios by leveraging machine learning techniques. The idea was inspired by the growing sophistication of network threats, Like the one Ransomware attack that happened at IIITD a few weeks back, and the potential of machine learning to provide better solutions in complex environments. We wanted to develop a robust model to predict and classify network intrusion. Find code here.

1. Problem Statement

Traditional security systems have become obsolete as advanced attacks cannot be detected in real time. The proposed project focuses on developing an advanced intrusion detection system based on the machine learning approach and includes Artificial Neural Networks and ensemble methods to classify network intrusions accurately in binary and in multi-class tasks with high accuracy.

Our Problem statement is to improve the detection of attacks such as DoS, Probe, R2L, and U2R, and also to address issues related to imbalanced classes and false alarms.

2. Literature Survey

We have reviewed and studied several research papers to gain insights regarding the already done work in the field and understand optimal data pre processing techniques and expected results. Here, we are mentioning the most relevant ones.

2.1. Paper 1

This paper proposes that the output layer of the architecture should be replaced by a linear SVM but with a margin-based loss function. The dataset was standardized and normalized and comprised of 4.1 GB of network traffic data. Results show that GRU-SVM performs superior to GRU-Softmax, achieving training and testing accuracies of 81.54% and 84.15%, better than Softmax (63.07%, 70.75%), while obtaining fewer false positives and negatives with similar runtime.

2.2. Paper 2

It is observed that various machine learning models are evaluated on network intrusion detection datasets. Comparison has been made among KDD CUP 99, NSL-KDD, and UNSW-NB15 using models such as Neural Networks, Support Vector Machines, and Random Forests, along with class balancing techniques that are SMOTE oversampling and random under-sampling. Results show significant superiority of the UNSW-NB15 and KDD CUP 99 compared to NSL-KDD dataset, particularly in the case of minority classes. Work further underlines the efficiency of F1 scores in assessment rather than accuracy and confirms that oversampling improves classification results.

2.3. Paper 3

This research explores machine learning-based NIDS against adversarial attacks. Methods include formulating the attack as a bi-level optimization problem with a GAN to generate adversarial features and devising a novel adversarial feature reduction defense method. Results showed that this proposed attack mostly evades NIDSs with a performance efficacy of over 97%. KitNET achieved an F1-score of 0.95 against botnet attacks with an evasion rate of up to 99.42%, and Support Vector Machines obtained a detection F1-score of 0.94, while the evasion rate stayed at 40.31%. The defense method proposed in the paper significantly enhances robustness by boosting F1 scores across classifiers.

3. Dataset

3.1. Class Evaluation

The dataset initially contained 22 intrusion classes, which were grouped into 4 broader categories: DOS, Probing, R2L, and U2R, along with 1 normal class. This grouping reduces complexity and addresses class imbalance. There are 40 feature columns, providing a wide range of attributes for analysis.

3.2. Feature Analysis

3.2.1 Derived and Traffic-Based Features

Time-based traffic features analyze connections to the same host or service over short intervals and are constructed us-

ing a window of 100 connections for attacks with longer intervals. Metrics such as ***count***, ***srv_count***, ***same_srv_rate***, and ***diff_srv_rate*** are key for detecting DOS, highlighting statistical trends and service-specific patterns.

3.2.2 Connection and Behavioral Features

Key features include ***src_bytes*** and ***dst_bytes***, which detect data exfiltration and Command & Control (C2) traffic. Features like ***protocol_type*** and ***service*** provide context, with uncommon combinations potentially revealing tunneling.

3.2.3 Error and Anomaly Indicators

Features like ***flag*** and ***wrong_fragment*** are indicators of scanning or probing. Binary indicators, ***land*** and ***urgent*** are suitable for immediate blocking rules. Sequential patterns in *flag* values often reveal network scanning activity.

3.3. Feature Interactions and Importance

Interactions between features like *duration* and ***src_bytes/dst_bytes*** provide insights into traffic efficiency, while combinations like ***count*** and ***error_rates*** measure attack intensity. Service-specific behavior can be profiled through metrics like ***service*** and ***srv_count***. Features such as ***src_bytes*** and ***duration*** exhibit extreme skewness, and authentication-related features show high sparsity.

3.4. Class Imbalance and Skewness

The dataset is highly imbalanced, with classes ranging from 52 to 391,458 instances. Techniques like SMOTEENN and SMOTENC are recommended for balancing. Features like *src_bytes* have extreme skewness ($\gamma_1 = 699.21$), necessitating transformations (e.g., log) for improved modeling.

3.5. Data Overview

The dataset contains 19,377 duplicate rows and shows high correlations in features. Careful handling of duplicates and feature interactions is critical for effective analysis.

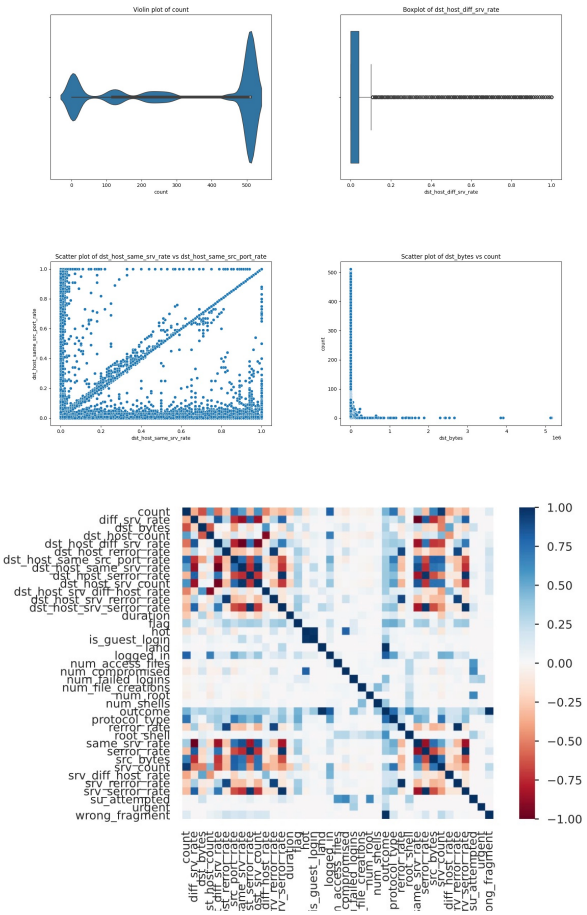
3.6. Implementation Recommendations

It is important to prioritize computationally efficient features and regularly update baselines while adjusting thresholds based on service needs. Time-series transformations and ratio derivations hold significant weight for feature engineering.

4. Methods and Models

4.1. Preprocessing Pipeline

The preprocessing pipeline ensures that the raw data is properly processed for training machine learning models, addressing class imbalance, encoding categorical features, scaling features, and performing feature selection.



4.1.1 One-Hot Encoding

Transforming categorical columns into binary matrices. This step is required so as to turn values in each category to a binary variable which helps in adding and better interpretation of the effect of that individual category Why preferred over label encoding? it is because the label has a naive assumption difference in the effect of labels 0 and 1 is the same as that of 1 and 2 and so on.

4.1.2 Feature Scaling

Robust scaling is applied to the dataset, which scales the data using the interquartile range (IQR). Feature scaling ensures that no individual feature disproportionately influences the model and minimizes extreme values' impact. Compared to standard scaling, affected by outliers, robust scaling focuses on the central part of the distribution and is more suitable for skewed data.

4.1.3 Resampling

The SMOTEENN (Synthetic Minority Over-sampling Technique + Edited Nearest Neighbors) method addresses

class imbalance by oversampling the minority class and undersampling the majority class. **Importance:** To deal with this class imbalance we have used SMOTEENN. It is an advanced variant of SMOTE. The technique works in two distinct phases. First, it applies SMOTE oversampling to the minority class where, for each minority data point, it identifies its nearest neighbors, randomly selects one of them, and creates a synthetic data point somewhere between these two points, adding it to the dataset. Then Edited Nearest Neighbor (ENN) undersampling is applied to the majority class. During ENN, it examines each data point by finding its nearest neighbors and removes the point if it belongs to the majority class and the majority of its neighbors belong to a different class.

4.1.4 Feature Selection

Firstly, highly correlated features are removed to reduce multi-collinearity. The remaining features are then evaluated based on their correlation with the target variable, and features with low correlation are removed.
Selected features: hot, count, srv_count, serror_rate, 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, protocol_type_icmp, protocol_type_tcp, service_ftp_data, service_http, service_other, flag_SF.

4.2. Models Methodology

We implemented several baseline models on our preprocessed data for evaluating the performance of future ensemble and artificial neural network (ANN) based models.

Decision Tree (J48) Our dataset contained more categorical columns than continuous ones. Decision Trees are particularly effective with categorical data as they can easily split on these features and capture complex patterns.

Perceptron We used Perceptron model as a simple baseline. It provides an initial understanding of how well the data can be separated using a linear decision boundary.

Naive Bayes It is a probabilistic classifier based on Bayes' Theorem, assumes independence among features. It performs well when features exhibit Gaussian-like distributions, as observed in parts of our dataset.

KNN classifies samples by considering the majority class of their k-nearest neighbors in the feature space. Useful for identifying local patterns, making it suitable for datasets with both categorical and continuous attributes like ours.

Random Forest combines the power of multiple Decision Trees through bagging, providing robustness against overfitting and enhancing generalization. Effective for datasets having diverse patterns and feature interactions.

Bagging classifier applies ensemble learning by training multiple estimators (e.g., Decision Trees) on bootstrapped subsets of the training data. By aggregating their predictions, it achieves higher accuracy and stability.

Boosting (AdaBoost) AdaBoost iteratively builds an ensemble of weak classifiers, focusing on misclassified samples to create a strong model. It is effective for capturing subtle patterns and addressing complex class distributions.

XGBoost XGBoost (Extreme Gradient Boosting) is an advanced ensemble technique that combines boosting with regularization. Known for its speed and efficiency, it optimizes both computation and predictive power, making it highly effective for handling large datasets with intricate patterns.

Softmax Regression A multiclass extension of logistic regression, this model helps understand how the data responds to linear classification within a probabilistic framework. It is useful for examining class probabilities and evaluating how well the data fits into distinct categories.

5. Results and Analysis

5.1. Graphs and metrices

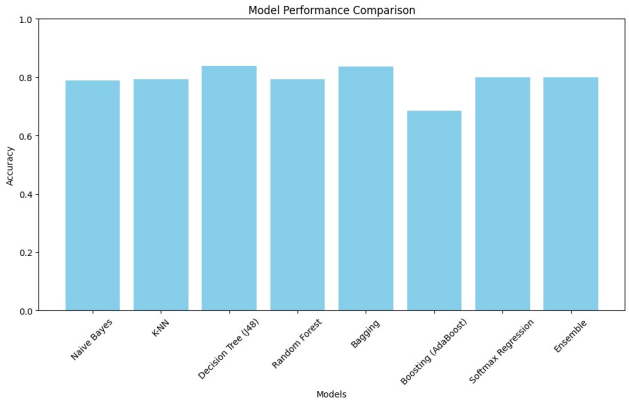
Model	Accuracy	Precision	Recall	F1 Score
Perceptron	0.7196	0.39	0.34	0.76
Decision Tree	0.7923	0.16	0.20	0.70
Softmax Reg.	0.7992	0.38	0.21	0.79
LightGBM	0.7924	0.16	0.20	0.70
Naive Bayes	0.7881	0.22	0.20	0.77
K-NN	0.7924	0.16	0.20	0.70
Dec. Tree (J48)	**0.8375**	0.36	0.25	**0.79**
Random Forest	0.7924	0.16	0.20	0.70
Bagging	0.8371	0.35	0.25	0.79
AdaBoost	0.6838	0.36	**0.37**	0.75
XGBoost	0.7923	0.16	0.20	0.70
Ensemble (Vote)	0.7988	0.36	0.21	0.73
K-NN (Tuned)	0.7924	0.16	0.20	0.70
Random Forest (Tuned)	0.7924	0.16	0.20	0.70
AdaBoost (Tuned)	0.7900	0.19	0.20	0.70
Neural Net (Tuned)	0.7700	0.22	0.20	0.77

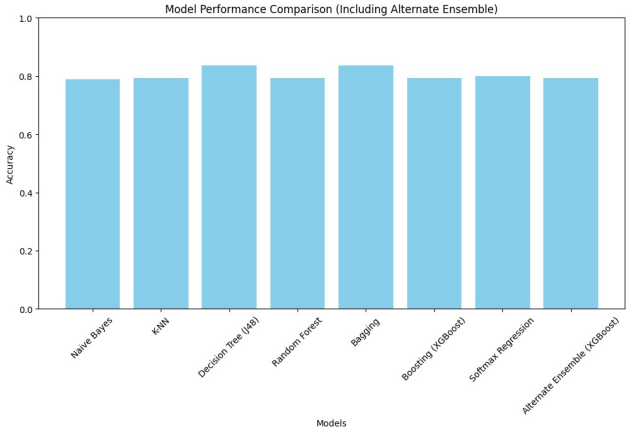
Table 1. Comparison of Models' Performance Metrics

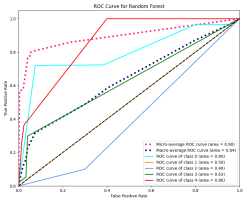
Algorithm Performance Analysis

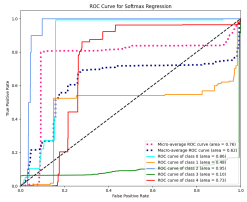
1. **Perceptron**: Struggled due to its linear nature, insufficient for complex non-linear boundaries, especially with class imbalance.

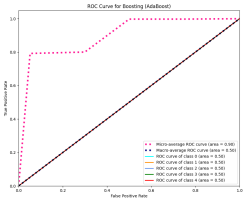
2. **Decision Tree**: Achieved high accuracy but overfit on the majority class, lacking generalization.

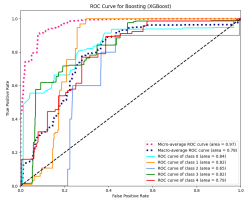












3. **Softmax Regression**: Handled multi-class classification better but suffered from class imbalance and non-linear data relationships.

4. **Naive Bayes**: Strong independence assumptions led to poor minority class performance but reasonable efficiency.

5. **K-NN**: Relied on majority-class neighbors, resulting in poor minority class recall despite high accuracy.

6. **Decision Tree (J48)**: Improved recall slightly due to ensemble-like structure, balancing high accuracy with better class representation.

7. **Random Forest**: Improved robustness through an ensemble of trees but remained focused on majority classes.

8. **Bagging**: Reduced variance compared to single trees, achieving high accuracy but minimal minority class gains.

9. **AdaBoost**: Focused on harder examples, improving minority recall slightly at the expense of overall accuracy.

10. **XGBoost**: Strong gradient boosting, but lacked specialized sampling to improve minority class performance.

11. **Ensemble (Voting)**: Majority voting leveraged multiple models' strengths but biased toward the majority class.

12. **Tuned K-NN**: Hyperparameter tuning refined results but couldn't overcome inherent majority-class bias.

13. **Tuned Random Forest**: Tuning improved tree depth but failed to address class imbalance effectively.

14. **Tuned AdaBoost**: Adjusted learning rate balanced focus on misclassified samples but remained weak for minority classes.

15. **Tuned Neural Network**: Hyperparameter tuning slightly improved generalization but suffered from overfitting and imbalance.

6. Conclusion

We implemented various machine learning models, such as Random Forest, Boosting algorithms, and others, to address the problem of network intrusion detection. Through this process, we learned and understood their workings and gained valuable insights by comparing their performances. One of the key findings was that Decision Trees were among the best models in terms of accuracy, demonstrating the algorithm's capability for creating effective separating boundaries. Major challenges included handling highly imbalanced classes within the dataset, which we addressed using the SMOTEENN technique. Another challenge was analyzing and selecting the best model among a vast number of algorithms. **Team Contributions:** **Angadjeet**: Model training and feature selection, **Apaar**: Model selection, literature review, and final evaluation, **Arnav**: Data preprocessing and model training, **Parth**: Architecture selection and model training. **Submitted By: Apaar**

References

1. Li, Z., Fang, W., Zhu, C., Song, G., & Zhang, W. (2023). Toward deep learning-based intrusion detection system: A survey. *ACM Computing Surveys*. Retrieved from https://dl.acm.org/doi/fullHtml/10.1145/3688574.3688578

2. Bibers, I., Arreche, O., & Abdallah, M. (2024). A comprehensive comparative study of individual ML models and ensemble strategies for network intrusion detection systems. *arXiv preprint*. Retrieved from https://arxiv.org/abs/2410.15597

3. Muhammad Ali, M., Haque, M., Durad, M. H., Usman, A., Mohsin, S. M., Mujlid, H., & Maple, C. (2023). Effective network intrusion detection using stacking-based ensemble approach. *International Journal of Information Security, 22*(4), 425–439. Retrieved from https://link.springer.com/article/10.1007/s10207-023-00718-7

4. Wardana, A. A., Kołaczek, G., Warzyński, A., & Sukarno, P. (2024). Collaborative intrusion detection using weighted ensemble averaging deep neural network for coordinated attack detection in heterogeneous networks. *International Journal of Information Security*. Retrieved from https://link.springer.com/article/10.1007/s10207-024-00891-3

5. Zhang, Y., Liu, J., Guo, X., & Wu, Z. (2020). A survey of intrusion detection techniques. *IEEE Communications Surveys & Tutorials*. Retrieved from https://ieeexplore.ieee.org/document/8999075

6. Kumar, S., Singh, A., & Sharma, P. (2024). A two-level ensemble learning framework for enhancing network intrusion detection systems. *IEEE Access*. Retrieved from https://ieeexplore.ieee.org/document/10540382

7. Wang, L., Liu, H., & Xu, J. (2023). Machine learning techniques for intrusion detection systems: A survey. *Applied Sciences, 9*(20), 4396. Retrieved from https://www.mdpi.com/2076-3417/9/20/4396

8. Arif, M., & Singh, K. (2024). A comprehensive survey on ensemble learning-based intrusion detection approaches in computer networks. *IEEE Transactions on Information Forensics and Security*. Retrieved from https://ieeexplore.ieee.org/document/10299619

9. Rashid, M., Arshad, S., & Khan, S. (2024). Machine learning and deep learning methods for intrusion detection systems: A survey. *Journal of Big Data, 10*, Article 109. Retrieved from https://journalofbigdata.springeropen.com/articles/10.1186/s40537-024-00886-w

10. Hanif, A., & Qureshi, R. J. (2022). Survey of intrusion detection systems: Techniques, datasets, and challenges. *Cybersecurity Journal, 7*(3), 234–252. Retrieved from https://cybersecurity.springeropen.com/articles/10.1186/s42400-019-0038-7

11. Cheng, K., Li, Y., & Wang, H. (2023). A neural network architecture combining gated recurrent units and a linear SVM with margin-based loss. *Papers With Code*. Retrieved from https://paperswithcode.com/paper/a-neural-network-architecture-combining-gated

12. Liang, J., Wang, J., Li, J., & Wang, H. (2018). Machine learning-based intrusion detection systems: An advanced survey. *arXiv preprint*. Retrieved from https://arxiv.org/pdf/1811.05372v1

13. Lin, W., Wang, W., Lu, X., & Chen, C. (2020). Adversarial machine learning in network intrusion detection systems. *arXiv preprint*. Retrieved from https://arxiv.org/pdf/2005.07519v4

14. Alomari, E., Abdullah, M., & Ismail, H. (2021). Review of intrusion detection systems: A comparative study. *Procedia Computer Science, 187*, 563–570. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050921011078

15. Kowalski, M., Kwiatkowski, M., & Pawlak, K. (2023). Network intrusion detection system: An innovative approach using AI. *Advances in Science and Technology Research Journal, 17*(3), 145–156. Retrieved from http://www.astrj.com/pdf-149934-76147?filename=Network+Intrusion.pdf

16. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408. https://doi.org/10.1037/h0042519

17. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324