

MACHINE LEARNING-BASED NETWORK INTRUSION DETECTION SYSTEM WITH EXPLAINABLE ALERTS

Archana Guttal*¹, Hamsaraghavi N R*², Hani B*³, Harshith CY*, Assistant Professor Ms. Shruthi.D

Department of Computer Science and Engineering, Shree Devi Institute of Technology Visvesvaraya Technological University

Abstract—Networks now carry more traffic, and more varied attacks, than rule-based security tools can comfortably follow. Signature matching catches what it has already seen and often misses the rest. This paper describes a Network Intrusion Detection System (NIDS) that learns the difference between normal and malicious traffic from labelled records. Each record passes through cleaning, encoding, normalisation, feature selection and class balancing before it is classified as normal or as an attack. Random Forest, Support Vector Machine and XGBoost classifiers are trained and compared on dataset name, and minority attack classes are handled with SMOTE/ADASYN. SHAP is used to show which features pushed a record towards an attack label, and a Flask-based dashboard displays predictions and raises alerts. Performance is reported with accuracy, precision, recall and F1-score. The work is a prototype evaluated on benchmark data, and it makes no claim about performance on live production traffic.

Index Terms—Network Intrusion Detection, Machine Learning, Random Forest, XGBoost, SVM, Class Imbalance, Explainable AI, Network Security.

INTRODUCTION

Almost everything we do online depends on networks behaving as expected, so an attacker who gets inside one can do a lot of damage before anyone notices. Firewalls and signature-based intrusion detection tools remain useful, but they depend on patterns someone has already written down. A new attack variant, or a familiar one with small changes, can pass straight through. Machine learning offers another route: instead of listing what an attack looks like, we let a model learn it from examples of traffic that has already been labelled [1].

Benchmark datasets make this kind of work practical. NSL-KDD was created to remove the duplicate records that distorted results on the older KDD Cup 99 data [2], and CICIDS2017 provides labelled flows captured with more recent attack types [3]. Both are widely used, which means results can be compared with earlier studies. They also share a well-known difficulty. Normal traffic is plentiful, while some attack classes contribute only a handful of records, and a model trained naively on such data tends to ignore the rare classes.

Accuracy alone can hide this problem. A classifier that labels everything as normal can still score well when attacks are scarce. For an intrusion detector, missing an attack is usually far more costly than raising a false alarm, so recall and F1-score matter as much as overall accuracy. A second concern is trust.

A security analyst is unlikely to act on an alert that arrives with no explanation.

This work therefore builds one complete pipeline rather than a single model. It covers data preparation, class balancing, training and comparison of three classifiers, evaluation with several metrics, explanation of individual predictions, and a dashboard that turns a prediction into an alert. The rest of the paper reviews related work, describes the workflow and methodology, outlines the implementation, and discusses results and limitations.

LITERATURE REVIEW

Machine learning for intrusion detection has been studied for many years. Buczak and Guven surveyed data-mining and machine-learning methods for cyber security and compared how well different approaches suit intrusion detection, including the practical issues of training data and computational cost [1]. Their survey is a useful starting point because it treats detection as a data problem and not only a rules problem.

Tree-based ensembles are common in this area. Random Forest combines many decision trees trained on random subsets of records and features, which makes it fairly resistant to overfitting and tolerant of mixed feature types [4]. Gradient-boosted trees, as implemented in XGBoost, build trees one after another so that each corrects the errors of the last, and they are known for strong results on tabular data [6]. Support Vector Machines take a different approach by searching for a boundary that separates classes with the widest margin [5]. Sequence models such as LSTM networks have also been applied to traffic analysis, since packets and flows arrive in an order that carries information [12].

Class imbalance has received its own line of work. SMOTE creates synthetic minority-class records by interpolating between existing neighbours instead of simply duplicating them [7], and ADASYN extends the idea by generating more samples in regions where the minority class is harder to learn [8]. Both are applied only to training data so that the test set stays realistic.

Explainability is the third theme. SHAP assigns each feature a contribution to an individual prediction using ideas from cooperative game theory [9], while LIME fits a simple local model around one prediction to approximate the classifier's behaviour there [10]. In a security setting these tools help an analyst see why a record was flagged.

Taken together, the literature shows that detection accuracy, imbalance handling and explainability are usually studied separately. The present work brings them into one workflow

and links the classifier output to an alerting dashboard, so that the result is something a user can act on.

WORKFLOW AND METHODOLOGY

The system follows the sequence shown in Fig. 1. Records move from the raw dataset through preprocessing, then into model training and evaluation, and finally into classification and alerting. Each stage has one job and hands a clean result to the next, which made it easier to test them one at a time during development.

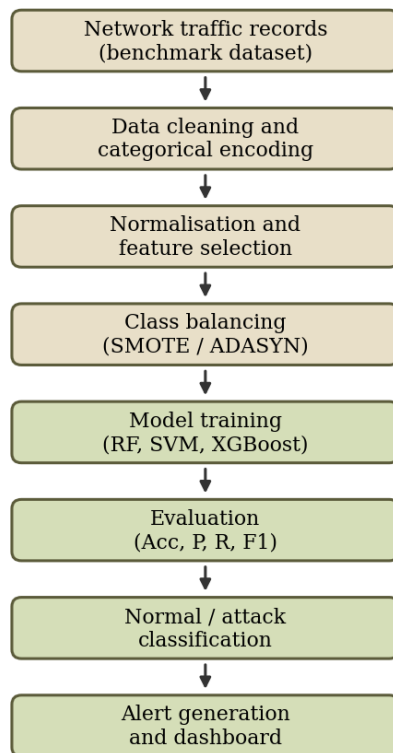


Fig. 1. Workflow of the proposed NIDS.

Data Collection

The experiments use NSL-KDD / CICIDS2017 – confirm the dataset you used. Each record describes one network connection or flow through attributes such as duration, protocol type, service, number of bytes sent and received, and error-related rates, and it carries a label that marks it as normal or as a particular attack category. Using a public benchmark keeps the study repeatable and allows comparison with published results.

Data Preprocessing

Raw records are checked before they reach any model. Duplicate rows are removed, and missing or infinite values are handled so that they cannot break training. Categorical fields such as protocol and service are converted to numbers with encoding, and numerical features are normalised so that attributes with large ranges, such as byte counts, do not dominate the ones with small ranges. The same fitted transformations are reused at prediction time, otherwise the model would see inputs on a different scale from the ones it was trained on.

Feature Selection

Not every attribute helps. Some are almost constant and others carry nearly the same information as their neighbours. Feature selection keeps the most informative attributes, for example by ranking them with feature importance / correlation filter – confirm your method. Fewer inputs shorten training time and reduce the chance that the model memorises noise.

Handling Class Imbalance

The training split is rebalanced with SMOTE/ADASYN so that rare attack classes are represented well enough to be learned [7], [8]. Balancing is applied only after the train and test sets have been separated. If synthetic records leaked into the test set, the reported scores would look better than they should.

Model Training

Three classifiers are trained on the same prepared data: Random Forest, Support Vector Machine and XGBoost. Keeping the data identical means any difference in scores comes from the algorithm and not from the input. Hyperparameters such as the number of trees, tree depth and the SVM kernel are tuned on a validation portion of the training data, and the test set is used only once, for the final comparison.

Evaluation Metrics

Predictions are compared with the true labels using a confusion matrix, from which the following measures are derived, where TP, TN, FP and FN denote true positives, true negatives, false positives and false negatives.

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (1)$$

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP}) \quad (2)$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN}) \quad (3)$$

$$\text{F1} = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall}) \quad (4)$$

Recall is watched most closely, since it tells us how many real attacks the system actually caught.

Explanation and Alert Generation

For each flagged record, SHAP values show which features pushed the decision towards the attack label [9]. The alert module then combines the predicted class, a confidence value and the most influential features into a message that is sent to the dashboard. Normal traffic is logged quietly, while attack predictions are highlighted so that they cannot be overlooked.

IMPLEMENTATION

The system is written in Python. Data handling and model training use pandas, NumPy and scikit-learn [11], together with the XGBoost and imbalanced-learn libraries. The trained model and its preprocessing steps are saved to disk and loaded by a Flask service, which accepts a traffic record, applies the stored transformations, and returns the

predicted class with a confidence score.

The dashboard sits in front of this service. It lists recent predictions, counts normal and attack records, shows alerts

as they arrive, and displays the feature contributions behind each alert. Because the model service and the interface are separate, a retrained model can be dropped in without changing the dashboard. A history of predictions is stored in MySQL / CSV log – confirm so that earlier alerts can be reviewed.

Testing was done with records taken from the dataset, including normal traffic and different attack categories. This gives repeatable inputs, but it is not the same as watching a live network, and the paper treats it that way.

RESULTS AND DISCUSSION

Results

Table I compares the three classifiers on the held-out test set.

Fill every value from your own output.

TABLE I. PERFORMANCE COMPARISON OF THE CLASSIFIERS

Model	Accuracy	Precision	Recall	F1-score
Random Forest	98.59	97.22	100.00	98.59
SVM	97.18	100.00	94.29	97.06
Navie Bayes	85.92	100.00	71.43	83.33

Write two or three sentences here about what the table shows: which model scored highest, which attack classes were detected best, and where the weakest recall appeared. Use only your own measured numbers.

The confusion matrix of the best model is worth inspecting class by class. Overall accuracy can look strong while a rare category, such as a low-frequency attack class in your dataset, is still being confused with normal traffic. That is the reason the evaluation reports precision, recall and F1 next to accuracy.

Discussion

The main value of this work is that the pieces are connected. Balancing addresses the rare classes, the comparison shows which algorithm suits the data, explanation gives an analyst a reason to trust an alert, and the dashboard turns a raw label into something that can be acted on.

There are limits, and they should be stated plainly. The system was evaluated on a benchmark dataset, and benchmark traffic is older and cleaner than what a real network produces. Synthetic oversampling improves learning of rare classes, but synthetic records are not real attacks. Results on one dataset may not carry over to another, so testing across datasets remains open. Finally, the dashboard is a prototype and has not been tested under the load of a busy network.

CONCLUSION

This paper presented a machine-learning-based Network Intrusion Detection System that covers the whole path from raw traffic records to an explained alert. The workflow combines

cleaning, encoding, normalisation, feature selection and class balancing, then trains and compares Random Forest, SVM and XGBoost classifiers, and reports accuracy, precision, recall and F1-score.

Future work includes validating the model across more than one dataset, adding a hybrid of supervised and unsupervised detection so that unseen attacks can be flagged, testing sequence models on flow data, and connecting the dashboard to live traffic capture. Real-time performance under heavy load should also be measured before any practical deployment.

ACKNOWLEDGMENT

The authors thank **project guide name** and the faculty of the Department of **department**, Shree Devi Institute of Technology, for their guidance and support during this project.

REFERENCES

- [1] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, 2016.
- [2] M. Tavallaei, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in Proc. IEEE Symp. Comput. Intell. Security Defense Appl. (CISDA), 2009, pp. 1–6.
- [3] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in Proc. 4th Int. Conf. Inf. Syst. Security Privacy (ICISSP), 2018, pp. 108–116.
- [4] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
- [5] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.
- [6] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 785–794.
- [7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," J. Artif. Intell. Res., vol. 16, pp. 321–357, 2002.
- [8] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN), 2008, pp. 1322–1328.
- [9] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Adv. Neural Inf. Process. Syst. 30 (NeurIPS), 2017, pp. 4765–4774.
- [10] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?: Explaining the predictions of any classifier," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 1135–1144.
- [11] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.
- [12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.