Anomaly Detection in Network Traffic

By Sawyer Blankenship, Anka Engin, Stephen Parker, Elijah Uy

Abstract

The goal of this project is to analyze the KDD Cup 1999 dataset and use machine learning models to detect anomalies in network traffic. We implemented two preprocessing methods: normalization and one-hot encoding. We also implemented four models: Random Forest, Neural Network, Decision Trees, and Support Vector Machine. Evaluation metrics included accuracy, precision, recall, and f1 score. The highest accuracy came from the SVM, which was 99.92%.

Introduction

The objective of this project is anomaly detection in network traffic. We trained various models on the KDD Cup 1999 dataset to classify network traffic data and compared their accuracy.

According to the UC Irvine KDD archive, "(The KDD Cup 1999 Dataset) is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment."

Our models classified traffic as normal, or as one of 37 types of malicious traffic, including buffer overflow, ipsweep, and mailbomb. The KDD dataset has 42 inputs including categorical, binary, and ordinal features.

Preprocessing

One-Hot Encoding

For the random forest and decision tree models, we applied one-hot encoding to categorical features. This preprocessing strategy ensured that each category was converted into its own feature dimension and helped to prevent any unintended ordinal relationships.

Normalization

To enhance our model's ability to understand our data, we will preprocess with normalization as well. Since we are building a neural network as one of our models, we need to

normalize numerical features to prevent exploding gradients. We will do this by using standard score normalization (z-score).

Model Implementation

Random Forest

The random forest model was implemented with using scikit learn's ensemble package. We used a RandomForestClassifier with Gini-Impurity loss fitted on the entire kddcup training dataset. For the random forest, we implemented a binary classifier by setting all y values to either 0 or 1, and the model predicted whether the connection was malicious. We also implemented a multiclass classifier using sklearn's Label Encoder which transformed label strings into a single integer value from 0 to the number classes.

Decision Tree

The decision tree model was also implemented with scikit-learn. Similarly we used Gini impurity as the criterion for splitting because it efficiently handles classification tasks by minimizing impurity at each node. The Decision Tree was trained on a preprocessed training dataset using the same binary classification approach as the Random Forest model. The hyperparameters were tuned using grid search cross-validation to identify the optimal depth enhancing the model's predictive accuracy and avoiding overfitting. A depth of 20 gave us optimal results.

Support Vector Machine

For implementation of the SVM, we first need to load the KDD Cup 1999 dataset. The dataset was accessed directly from an online .gz file using pandas, eliminating the need for manual downloads or decompression. After loading, categorical features such as protocal_type, service, and flag were encoded numerically using LabelEncoder to make them suitable for model input. The target variable label is simplified into a binary classification problem to distinguish from normal and attack traffic. The input features x and label targets y are then separated, and all numerical features are scaled using StandardScalar to normalize the ranges.

Once the data was preprocessed, it was split into training and testing sets using a 70/30 split to train the model and evaluate it on unseen data. A Support Vector Classifier (SVC) with a Radial Basis Function (RBF) kernel was then initialized. The kernel allowed the model to capture non-linear relationships with the data, which is essential for distinguishing complex patterns of normal vs anomalous behavior in traffic. The model was trained on the scaled training data using .fit().

Neural Network

The Neural Network was implemented to have an in depth training process. We used two preprocessing methods, one-hot encoding and normalization using z-score standardization, which prevents issues like exploding gradients. After formatting the data and splitting them into training, testing, features and labels, and preprocessing the data, we trained the data. We used a tanh activation function in the hidden layer, and a sigmoid function at the end for binary classification for an in-depth forward propagation. We used the derivatives of these functions for an in-depth back propagation. After running through 200 iterations, the neural network resulted in a precision of 99.5%.

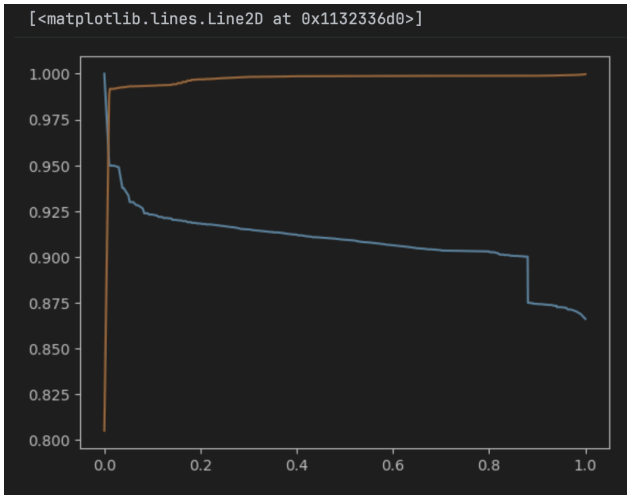
Evaluation and Analysis

Random Forest

We tested the random forest on kddcup's labeled dataset in corrected.gz. We noticed that our training dataset(kddcup.data.gz) only had 23 possible labels, while our testing dataset(corrected.gz) had 38 possible labels. Despite this, our model still achieved excellent results.

For our random forest, we used scikit's predict_proba function which estimates probabilities for each class. We wanted to emphasize recall over precision because the cost of being alerted to a benign connection is much less than the potential consequences of not detecting a malicious connection. The predict_proba function predicted 1.000 or 0.000 for many connections, so even when we flagged all y-hat values with probabilities >= 0.00000001 as malicious, 19.48% of test inputs were classified as benign.

This graph shows how the precision(orange) and recall(blue) relate to the threshold.



With the threshold as 0.00000001, our binary classification random forest classified 95.61% of test examples correctly. 95.11% of malicious test examples were correctly identified(recall) and 98.8% of examples flagged as malicious were

actually malicious(precision). The recall seems to be capped at 95.11% (unless we classify all examples as malicious). We suspect this low recall is because 15 types of malicious network activities in the test dataset were not exemplified in the training dataset.

We also trained and tested a multiclass random forest classifier. After training, our model correctly predicted the label for 91% of test examples. This is better than expended considering 80.52% of test examples were malicious and our test dataset had 37 different malicious classes.

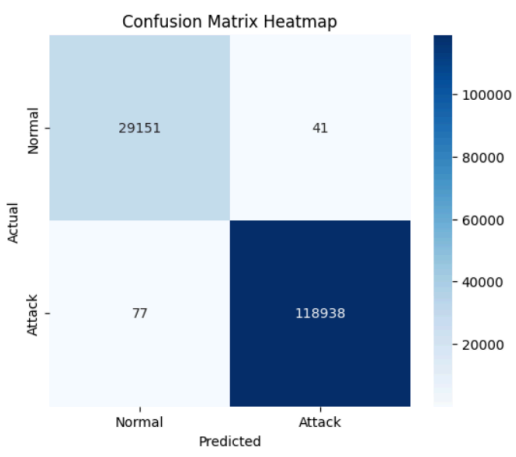
Decision Tree

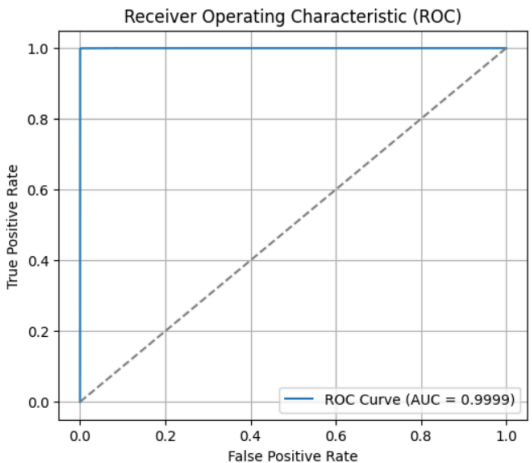
The Decision Tree was evaluated using a similar criteria to the random forest and the same data set. The model managed to achieve an overall accuracy of roughly 99%. The decision tree executed very quickly compared to the other ensemble methods making it suitable for environments where real-time analysis is necessary. We also observed that the Decision Tree was more likely to overfit when compared to the Random Forest with a very high precision and recall. We fine tuned the model and tested that it was not overfit to combat this.

SVM

After training the SVM model, its performance is evaluated using the test dataset. The predictions generated by the model are compared to the actual labels using a confusion matrix, which provides a clear view of how well the model distinguishes between normal and attack traffic. The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives, which are then used to calculate key classification metrics. The classification_report from scikit-learn outputs precision, recall, F1-score, and overall accuracy. These metrics offer a detailed understanding of the model's ability to correctly identify both normal and anomalous activity.

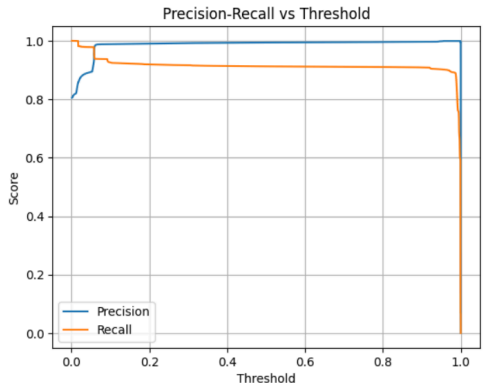
In the results, the model achieved exceptionally high precision and recall values for both classes, indicating that it was highly effective at detecting attacks while minimizing false alarms. However, while the classification report initially shows an accuracy of 1.00, further inspection of the confusion matrix reveals that the true accuracy is slightly lower (~99.92%), with a small number of misclassifications. Visual tools such as a confusion matrix heatmap and a ROC curve are used to further interpret performance. The ROC curve, with an Area Under the Curve (AUC) close to 1.0, confirms that the model maintains a strong balance between sensitivity and specificity, making it a reliable tool for intrusion detection in network environments.





Neural Network

The Neural Network's performance was evaluated based on accuracy, precision, and recall. While it did not achieve the highest accuracy overall (91.93%), it demonstrated strong generalization with an F1-score of 0.9949 and ROC-AUC of 0.9968, which showcases an excellent balance between precision and recall. If the Neural Network had a few more hidden layers, or had run for a few more epochs, perhaps it would've had a better accuracy score.



Comparative Analysis

Model	Accuracy	F1 Score	ROC-AUC	Train Time
Random Forests	0.956	0.969	0.9996	10 minutes
Decision Trees	0.99	0.9996	0.9997	30 s*
Neural Network	0.9193	0.9949	0.9968	5 s*
Support Vector Machine	0.9992	0.9992	0.9999	32 s*

*Trained on 10% subset of Data

Conclusion

Out of the four models used, SVM had the highest accuracy score (99.92%), and it demonstrated expert recognition of normal and malicious traffic. If it had run for more iterations, the neural network might've had better precision, as it had an effective preprocessing pipeline. For future work, more advanced architectures could be implemented like deep learning or recurrent networks.

Individual Contributions

Sawyer - Intro, One hot preprocessing, Random Forest Model and Evaluation
Anka - Normalization preprocessing, Comparative Analysis, Neural Network Model and Evaluation, Abstract, Conclusion
Stephen - Decision Tree Model and Evaluation
Elijah - Support Vector Machine Model, Comparative Analysis, and Evaluation

Works Cited

KDD Cup 1999 Data, 18 Oct. 1999, kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.