ICS 635 Final Project

Fahim Yasir

May 10, 2024

This is the project report for the final project of the 'ICS 635: Machine Learning' course, where different machine learning (ML) techniques have been used to design a classifier.

Problem Introduction:

The problem being addressed involves the increased necessity for network security due to the growth in computer network usage and the vulnerabilities in systems that could lead to malicious attacks. Traditional rule-based security measures are often inadequate against sophisticated cyber threats. The aim is to create a model that can accurately and promptly determine the presence of an attack within network packets, thus providing a more effective defense mechanism in the realm of cybersecurity.

The inputs to the model are the 41 features per record derived from network traffic data. These features include time-based traffic features, host-based traffic features, and content features. These inputs capture different aspects of network behavior and potential security threats. The output of the model is a label that categorizes each connection as either normal or an attack. The function being approximated by the ML models is a mapping from the 41 input features to a classification label that indicates whether a network connection is normal or an attack. The models essentially act as a classifier that analyzes patterns in network traffic to detect and categorize potential security threats.

ML is particularly appropriate for this task due to its ability to learn from large amounts of data and generalize from those to detect both known and unknown types of threats. ML-based methods can adapt to new threats by learning from anomalies in network behavior that do not match the established normal patterns. The training dataset for this project consists of 125,972 samples, providing a substantial amount of data for training the models. This sizeable dataset allows the models to learn a comprehensive range of patterns and behaviors, increasing its effectiveness in real-world scenarios. Other methods, such as signature-based detection systems, fail in scenarios involving new or evolving malware because they rely on a fixed database of known patterns. They cannot adapt to new threats that do not match these pre-defined signatures, making them less effective against novel attacks.

Project Motivation:

I chose this project primarily for two reasons: firstly, my interest in the data security domain, and secondly, I felt this dataset was well-suited for all the techniques discussed in this class. The dataset is quite large and features a variety of characteristics. Previous work on this project has primarily focused on the networking aspects and other solution techniques besides machine learning. In contrast, work that did focus on ML was not extensive, often concentrating on a single type of model or mainly on data analysis.

This is why I saw this project as an opportunity to build an almost perfect classifier using all the techniques taught in this course, aiming to develop a comprehensive solution to this data security issue.

Dataset Introduction:

The dataset used for this project originates from the 1998 DARPA Intrusion Detection Evaluation Program, managed by MIT Lincoln Labs. This program was designed to evaluate research in intrusion detection within a simulated military network environment, specifically mimicking a typical U.S. Air Force LAN. The raw training data comprises approximately four gigabytes of compressed binary TCP dump data collected over seven weeks of network traffic, which was processed into about five million connection records.

The dataset used for this project is a refined subset of the original data, includes 125,972 rows and 43 columns. It eliminates redundancies found in the original dataset, ensuring that the classifiers are not biased toward more frequent records. Each record in the dataset, which serves as input, is described by 41 features. These include time-based traffic features, host-based traffic features, and content features that capture various aspects of network behavior and potential security threats. The remaining two columns include one that serves as the label for each record, categorizing each connection as either normal or an attack, which is the output of the model. The other column was not relevant to the feature set and was therefore dropped from the analysis. This setup ensures that the model is trained on a diverse and comprehensive dataset, essential for developing a robust ML model.

Sample data points from the dataset have been shown in the Notebook of this project.

Model Introduction:

As mentioned previously, I wanted to use this project as an opportunity to apply all the machine learning techniques learned in this course to develop a robust classifier. That's why I used a variety of models, starting from a basic regression model to a neural network. The models used include: Logistic Regression (LR) and Gaussian Naive Bayes (GNB) as the baseline models, K-Nearest Neighbors (KNN) to observe the performance with different numbers of neighbors, Decision Tree (DT) and Gradient Boosting (GB) Classifiers, since ensemble techniques generally perform well, as discussed in the lectures. Another ensemble method, Random Forest (RF), was used to assess the effects of Principal Component Analysis (PCA). Support Vector Machine (SVM) was chosen to examine the impact of different Kernels. A Voting Classifier was also tested to see if a combination of ensemble models yields better results than a single ensemble model. Lastly, a Neural Network (NN) was used to compare the performance between machine learning and deep learning techniques.

Features & Pre-Processing:

As mentioned earlier, the dataset contained 41 features divided into three categories: basic features of individual connections, content features within a connection, and traffic features. No units were specified for the features in the dataset. The feature types are a mix of

continuous and discrete, and the data types include integers, floats, and objects (for categorical features). One-hot encoding was applied to the categorical features to make them suitable for the machine learning models. However, the number of unique categories for one of the categorical features was greater in the training set than in the test set. This discrepancy led to a mismatch in the sizes of the encoded test and train datasets. To resolve this issue, I added the missing columns to the test set and filled them with zeros. I found this solution on a StackOverflow discussion, where other solutions were also suggested, such as combining both datasets, applying one-hot encoding, and then splitting them. However, this approach seemed more sensible and convenient to me, which is why I chose it. There were no missing or duplicated records, so no specific actions were required in that regard. Lastly, the Robust Scaler method was used to scale the features, making them robust to outliers.

Model Evaluation:

For this project, the main dataset of 125,972 examples was randomly shuffled and split into two subsets: 75% for training and 25% for validation. I also implemented 10-fold cross-validation (CV) specifically for DT and GB models during hyperparameter tuning. Models were trained on the training set, while the validation set was used for early stopping, hyperparameter optimization, and model selection.

The test dataset, which is completely separate and consists of 22,543 examples, includes unique attack types not present in the training data to ensure realistic testing conditions. This test set was used to evaluate the final model.

Hyperparameter Search Space:

For this project, I used a number of different models, each with a distinct hyperparameter search space. I tried to choose the most important yet varied parameters for each model to see how changes in these parameters affect the model's performance.

Starting with the baseline models, for LR and GNB, I trained 1 model each using default parameters set by Scikit-learn (Version 1.2.2); no hyperparameter values were explored for these two models. However, from KNN to NN, different hyperparameter values were searched for each model. For KNN, I tested all odd values of K from 1 to 10, resulting in the training of 5 models. For DT, I explored tree depth and maximum features from 1 to 9 using 10-fold CV, which led to the training of 810 models. For GB, I searched the tree depth from 1 to 10, number of trees between 50 and 150, and learning rates from 0.01 to 0.2, {0.01, 0.05, 0.1, 0.2}, training a total of 100 models using 10-fold CV. Next, for RF, I did not search for any optimal hyperparameters; instead, I tried different numbers of components for PCA, {10, 15, 20}, and applied RF with PCA, resulting in the training of 4 models (including 1 on the original dataset). Then, for SVM, I experimented with different kernels: linear, polynomial, RBF, and sigmoid, leading to 4 models being trained. Lastly, for NN, I tried different numbers of layers from 3 to 7, neurons from 64 to 512, various activations like relu, sigmoid, tanh, and different dropout rates from 0.2 to 0.7, training approximately 100 models.

Hyperparameter Optimization:

As mentioned in the previous section, I used different models for this project, each with different hyperparameter search spaces. This project was considered an opportunity to apply various machine learning techniques, which is why I used different optimization methods for each model to better understand how each technique works and to determine which is most suitable.

For the baseline models, LR and GNB, I did not apply any hyperparameter optimization. For KNN, I iterated through different values of K and recorded the accuracy on both the training and validation sets. The model with the highest accuracy was then tested on the test set. For DT, I used the Grid Search technique with 10-fold CV. After determining the best parameters from the Grid Search for DT, those parameters were used to build the final DT model. For GB, I used the same approach but opted for Random Search instead of Grid Search. For RF, as previously mentioned, I did not use any optimization technique for the model itself, specifying a tree depth of 9. However, for PCA, I experimented with different component values to see how PCA affects performance and to compare the performance of RF on the original dataset versus the PCA-reduced dataset, using AUROC values as the metric. For SVM, I fixed all parameters except the kernel type to determine which SVM configuration worked best for my model. Lastly, for NN, I used a fully connected architecture with the 'adam' optimizer, and manually tuned parameters such as the number of layers, neurons, activation functions, and dropout rates, monitoring the accuracy and loss on the validation set to select the best model, which was then evaluated on the test set.

The following table will provide a summary of the hyperparameters that were tuned, the search space, the optimal values, and the number of models I trained.

Model	Hyperparameters	Search Space	Optimal Value	Model Trained
K-Nearest Neighbors	K	{1, 3, 5, …, 9}	3	5
Decision Tree	Tree depth Maximum features	{1, 2, 3, …, 9} {1, 2, 3, …, 9}	9 9	810
Gradient Boosting	Tree depth Number of trees Learning rates	{1, …, 10} {50, …, 150} {0.01, 0.05, 0.1, 0.2}	9 100 0.05	100
Random Forest with PCA	Component of PCA	{10, 15, 20}	N/A	3
Support Vector Machine	Kernel	{Linear, Poly, RBF, Sigmoid}	Linear	4
Neural Network	Layer Neurons Activation function Dropout rate	{3, 4, …, 7} {64, 128, 256, 512} {relu, sigmoid, tanh} {0.2, 0.3, …, 0.7}	5 varies varies 0.4	100 (approximately)

Apart from the models mentioned in this table, four other models were trained: one LR, one GNB, one RF on the original dataset, and one Voting Classifier.

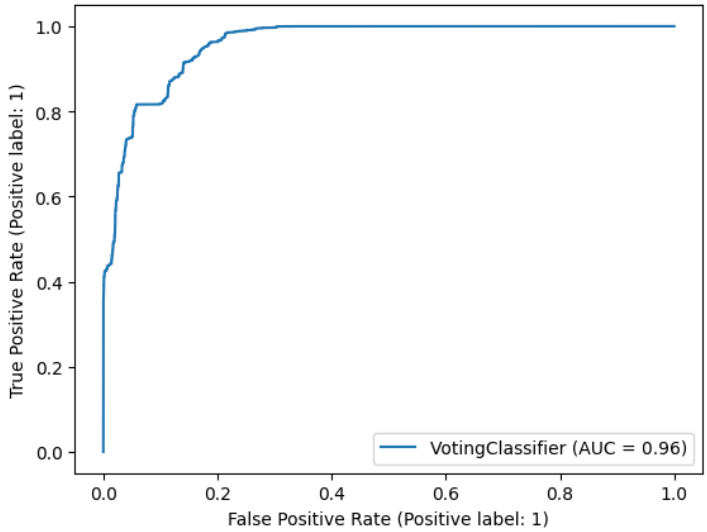
For the models I hand-tuned, such as NN, I did not follow a systematic approach. I frequently changed parameters, tried different model architectures, and checked their performance. Therefore, the numbers I mentioned for these models might be slightly inaccurate, either higher or lower.

Another thing I should mention is that exploring more values for KNN and GB could have potentially increased the accuracy. However, training these models was taking too long, which is why I explored only a small set of numbers for these models. It might be possible to achieve better accuracy if I could try more model parameters, but it wasn't time-efficient.

Evaluation on Test Set:

The best performance on the test set, an AUROC of 0.96, was achieved by the ensemble techniques, specifically GB and Voting Classifier. The AUROC curve for the Voting Classifier is shown below, and the AUROC values for all other models on the test set are presented in the following table.

(The AUROC curves for the final model of each type are shown in the Notebook.)



Model	AUC
LR	0.88
GNB	0.90
KNN	0.83
DT	0.90
GB	0.96
RF	0.95
RF with PCA	0.91
SVM	0.84
Voting Classifier	0.96
NN	0.92

It is evident from the table that ensemble methods are the most effective technique for solving this problem.

Distribution Shift Risk:

The model's robustness against distribution shift was tested by including unique attack types in the test dataset that were not present during training. While the model's design aims to ensure good generalization to new threats, it's important to note that changes in attack patterns could still affect performance. To counteract potential degradation, continuous monitoring and periodic updates with new data are recommended.