Matthew Poncini
*J.B. Speed School of Engineering*
*University of Louisville*

**23 October 2025**

—

# Project 2:
# Classification of a Trojan Horse Data Set

**Abstract**

This project develops and evaluates six binary classifiers: Logistic Regression, Decision Tree Algorithms, Naïve Bayes, and K-Nearest Neighbors KNN. These classifiers are used to analyze a web traffic dataset containing Trojan horse attacks. Using the results from the top performing classifiers, an ensemble stacking classifier was created to give the final predictions. The paper outlines the complete data mining process, including data preprocessing and feature reduction. It then examines the performance of the three classifiers in distinguishing "trojan" from "benign" network traffic. Following an analysis of preliminary results, limitations are identified, and improvements to the classification pipeline are explored. The paper also presents background on web traffic attack patterns and summarizes related academic research on data mining techniques in cybersecurity.

# Data Mining Project: Classification of a Trojan Horse Data Set

## 1. Problem Description

Trojan horse attacks are very prevalent on modern networks. This malware disguises itself as legitimate software, while secretly performing actions in the tunes of data exfiltration, backdoor access, or keylogging. Trojans typically bypass traditional signature-based defenses, which makes behavioral and traffic-based analysis critical for detection. Because trojan behaviors are subtle in network flow, machine learning models can detect statistical deviations from benign traffic patterns. One of the challenges of classifying web traffic data to *trojan* or *benign* is that there is high dimensionality and high feature correlation.

In this project, we are provided a dataset with 85 features with 1 target column class: *trojan* or *benign*. The dataset provided has nearly 160k instances of web traffic, with "Class" being divided nearly equally. The objective of this study is to develop and evaluate predictive models capable of accurately classifying web traffic instances into their correct categories based on the statistical features of network flows.

## 2. Review of Publications

In preparation for this project, three publications were used to provide a basis for how data mining techniques can be applied to network traffic for trojan horse detection.

### 2.1 *Machine Learning for Encrypted Malicious Traffic Detection: Approaches, Datasets and Comparative Study*

Zihao Wang, Kar Wai Fok, and Vrizlynn L. L. Thing present their paper as a survey of machine-learning approaches to encrypted malicious traffic detection. They identify that as encryption becomes more popular for privacy and security, adversaries use encrypted malicious traffic to blend their tracks. The paper presents this behavior as untraceable with traditional inspection techniques such as deep packet inspection. They compare previous studies that use different datasets to make a framework for machine-learning-based detection of encrypted malicious traffic. After creating a standard dataset, they compare the traditional machine learning approaches with deep learning approaches. In their experiment, they presented the results of the top ten performing algorithms among five different feature sets: Random Forest (RF), XGBoost, C4.5, AdaBoost, CART, K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Naïve Bayes, Logistic Regression, and Linear Regression. They found that ensemble-based methods such as RF (accuracy $\approx 0.95$) and XGBoost (accuracy $\approx 0.94$) achieved the highest overall detection performance on encrypted malicious traffic. Simpler classifiers like Logistic Regression and Naïve Bayes performed notably worse, highlighting the advantage of nonlinear ensemble approaches for complex network-flow behavior. Based off the research studies they examined, they concluded that the quality of the dataset used was the biggest driver in the performance of that model.

2.2 *Network traffic classification: Techniques, datasets, and challenges*

This publication, written by Ahmad Azab, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour, is a survey of methods for identifying and categorizing network traffic by application, protocol or service. They organize prior research into four major categories: port-based, deep-packet-inspection, machine-learning, and deep-learning methods. They outline the machine learning pipeline: data collection, flow representation, feature engineering,datasets preparation, model building, and model evaluation. They summarize the following supervised machine learning techniques: SVMs, Naïve Bayes, C4.5, C5.0, Random Forest, and K Nearest Neighbors. They further emphasize how deep learning models, including CNNs, RNNs, and autoencoders, have recently gained traction for traffic classification due to their ability to automatically extract hierarchical representations from raw or minimally processed network flows. These models eliminate much of the manual feature engineering required by traditional machine learning approaches, though they demand larger labeled datasets and substantial computational resources. The paper identifies encryption as the most significant barrier to accurate classification. The authors also note additional obstacles such as class imbalance, evolving application behavior, and concept drift in real-world network environments. They propose several future research directions including the creation of benchmark datasets that better reflect modern encrypted traffic, hybrid models that combine statistical and deep learning features, and lightweight classification frameworks suitable for real-time deployment.

2.3 *Trojan Traffic Detection Based on Meta-learning*

Zijian Jia, Yepeng Yao, Qiuyun Wang, Xuren Wang, Baoxu Liu and Zhengwei Jiang address the problem of detecting traffic generated by trojan malware in real-network settings where only a small number of traffic samples may be available. They note that conventional deep-learning or machine-learning methods usually require large volumes of labeled training data, which is at times impossible to collect quickly when a new variant of the malicious software appears. They propose a meta-learning framework composed of three parts. The first part is the embedding part, which "aims to extract the original Trojan traffic samples, transform them into computable feature vectors, and then divide the model into meta tasks." The embedding network combines a Residual Network (ResNet) and a Bidirectional Long Short-Term Memory (BiLSTM) model to capture both spatial and temporal features of traffic flows. The relation network is enhanced with dynamic routing inspired by capsule networks, which improves similarity computation between samples and class prototypes. The second part is the relation network based on metric. They formulated detection as a C-way K-shot problem, meaning the model learns across multiple small “meta-tasks,” each containing C classes and K support samples per class, enabling the system to generalize from very limited examples of new Trojan families. The third part is verification. The meta-learning model achieved higher accuracy, precision, and F1 scores compared to standard CNN, BiLSTM, and ResNet baselines, particularly in few-shot conditions. For example, it reached approximately 0.857 precision, 0.829 recall, and 0.842 F1-score, outperforming conventional classifiers that struggled with limited data. Their results show that the meta-learning approach is promising in for Trojan traffic detection in scenarios of limited labeled data and class imbalance.

## 3. Software tools

The primary software tool used for this project was the Python ran on a Jupyter Notebook. All techniques used in the predictive data mining section of this project came from the Scikit Learn libraries.

Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression

LogisticRegression(
    penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True,
    intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs',
    max_iter=100, multi_class='deprecated', verbose=0,
    warm_start=False, n_jobs=None, l1_ratio=None
)
```

C4.5 and C5.0 Decision Tree Classifiers

```
from sklearn.tree import DecisionTreeClassifier

DecisionTreeClassifier(
    criterion='entropy', splitter='best', max_depth=None, min_samples_split=2,
    min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
    random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
    class_weight="balanced", ccp_alpha=0.0, monotonic_cst=None
)
```

Naïve Bayes (Gaussian) Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
acc_nb = accuracy_score(y_test, y_pred_nb)
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

RandomForestClassifier(
    n_estimators=100, criterion="gini", max_depth=None, min_samples_split=2,
    min_samples_leaf=1, max_features="sqrt", bootstrap=True, oob_score=False,
    random_state=None, n_jobs=None
)
```

K-Nearest Neighbors (KNN) Classifier

```
from sklearn.neighbors import KNeighborsClassifier

KNeighborsClassifier(
    n_neighbors=5, weights="uniform", algorithm="auto", leaf_size=30,
    p=2, metric="minkowski", metric_params=None, n_jobs=None
)
```

All code shown follow the official examples provided in the scikit-learn.org documentation and public educational resources. These code templates were adapted to the trojan horse datasets for this project.

## 4. Preprocessing

For preprocessing, the following actions were taken: identify and handle missing data, features outlier detection and handling, normalization, cluster outlier detection and handling, and dimensionality reduction.


Figure 1: *Preprocessing pipeline*

### 4.1 *Identify and Handle Missing Data*

The first action taken in preprocessing was treating all -1 values as missing values. This was justifiable as negative values weren't appropriate for any of the features and this is a common input for missing data. The second action was to count the number of missing data for each feature.
With nearly half of the data missing for the Init_Win_bytes_backward feature, there was a large enough of a compelling case to drop this feature. Keeping this feature would inject too much uncertainty and noise. By

Table 1: *Missing Values by Feature*

| **Column Name** | **Missing Values** | **% Missing** |
|---|---|---|
| Total Backward Packets | 2,554 | 1.60% |
| Total Length of Fwd Packets | 2,764 | 1.73% |
| Fwd Packet Length Max | 3,555 | 2.23% |
| Fwd Packet Length Min | 4,535 | 2.84% |
| Flow IAT Min | 121 | 0.08% |
| Bwd IAT Mean | 192 | 0.12% |
| Fwd Header Length | 6,092 | 3.81% |
| Bwd Header Length | 23 | 0.01% |
| Fwd Header Length.1 | 26 | 0.02% |
| Fwd Avg Bytes/Bulk | 3,747 | 2.35% |
| Init_Win_bytes_forward | 34,955 | 21.88% |
| Init_Win_bytes_backward | 77,162 | 48.31% |
| min_seg_size_forward | 54 | 0.03% |

dropping Init_Win_bytes_backward, it is possible that we will prevent misleading correlations. For the rest of the missing data, two actions were taken. The first action was to add a feature for each of the features that contained missing data. The added feature flagged whenever a row had a missing data point for a feature. This preserves a potentially important signal; if certain flows tend to omit a header field in Trojan traffic but not in benign traffic, the model can learn that pattern. The second action was to impute the column average for each missing value. Mean imputation keeps the variable's overall distribution stable and allows prediction algorithms that cannot handle missing data to train properly. By adding the missing feature flags, bias introduced by imputing means is partially corrected.

4.2 *Features Outlier Detection and Handling*

Outliers were detected using the standard deviation method. This measures how far each observation deviates from the mean in units of standard deviation. For each numeric feature, any value that was than greater than three standard deviations away from the mean was considered an outlier.

Following detection, a two-step treatment was applied similar to the missing data procedure. First, a new binary indicator column was created for each feature, flagging rows that contained an outlier value. Second, the outlier values themselves were replaced with the column mean to reduce the influence of extreme magnitudes. This approach maintains the integrity of the dataset while minimizing the distortion that outliers can introduce into scaling and model training.

4.3 *Normalization*

For this data set, Min-Max Normalization was used. This ensured that all numeric data fell between 0 and 1. Normalization is essential for algorithms that are sensitive to magnitude differences between features. Min-max normalization preserves the shape of each distribution while unifying the scale.

4.4 *Cluster Outlier Detection*

Cluster-based outlier detection was performed using the BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm, following the methodology outlined in the article "Anomaly Detection with BIRCH" by Abhishek Biswas. The approach builds a hierarchical clustering structure where each sub-cluster center represents a typical pattern in the feature space. After fitting BIRCH on the training data, the Euclidean distance from each observation to its nearest sub-cluster center was computed. By applying a threshold set at the 99th percentile of the training distances, flows that exceeded this cutoff were flagged as outliers and assigned a binary indicator.

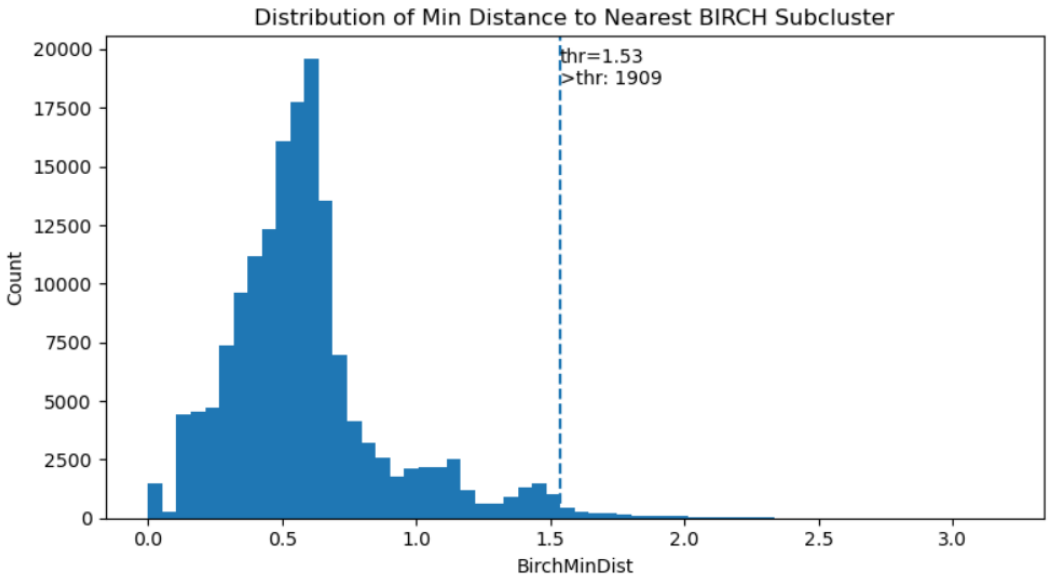


Figure 2: *Distribution of minimum distances from each observation to its nearest BIRCH subcluster center. The dashed vertical line marks the 99th-percentile threshold, above which approximately 1,909 instances were flagged as outliers.*

### 4.5 *Dimensionality Reduction*

The first features that were dropped were excluded because they were non-numeric identifier fields that did not carry meaningful statistical information about network behavior. The following features were dropped: "Flow ID", "Source IP", "Destination IP", "Timestamp", and "Unnamed: 0".

Principal Component Analysis (PCA) was selected as the dimensionality reduction technique due to its ability to capture the maximum amount of variance in the data using a smaller set of uncorrelated components. The original feature set contained many interdependent network metrics, where strong correlations among flow-based variables can obscure the underlying data structure. PCA addresses this by projecting the data into a new orthogonal feature space, where each principal component represents a linear combination of the original features that explains a decreasing proportion of total variance. This transformation not only simplifies the feature space but also enhances model efficiency by reducing redundancy and noise.

The dataset contained 98 numeric features describing network flow behavior (features had been added to flag missing and outlier data). Principal Component Analysis was applied to reduce the dimensionality while preserving most of the dataset's information content. The feature set was reduced to 28 principal components, which accounted for approximately 99% of the total variance. This demonstrates that nearly all meaningful variability in the data could be represented within a much smaller feature space. The resulting components capture the dominant correlations among features while minimizing redundancy and noise, providing a more efficient and interpretable representation of the dataset for subsequent classification modeling.

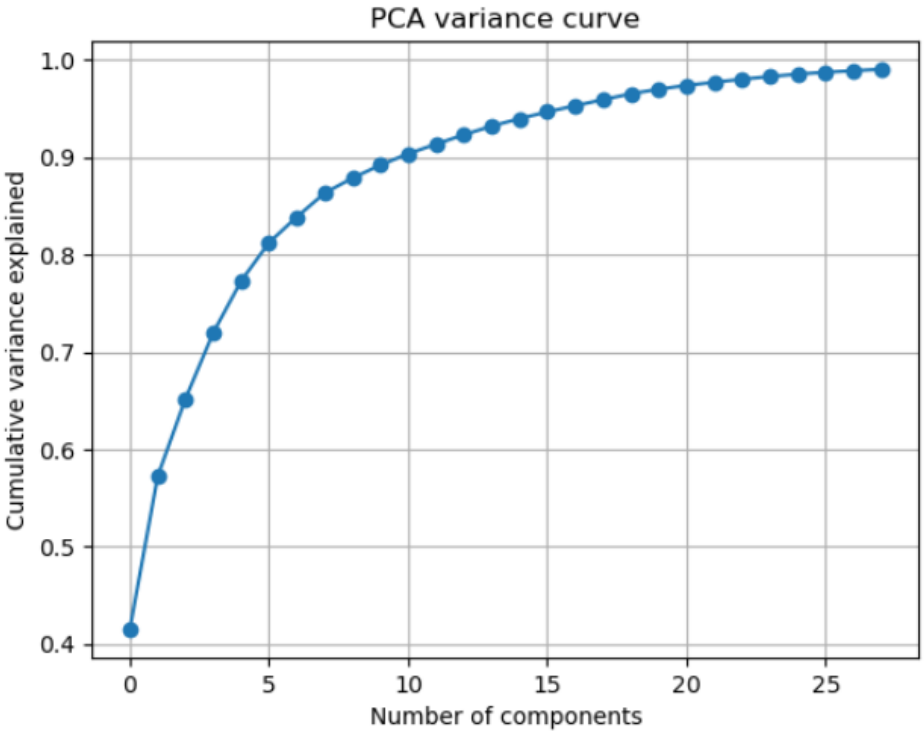


Figure 3: Cumulative variance explained by the principal components. The curve shows that the first few components capture the majority of the dataset's variance, with diminishing returns beyond approximately 20 components. Retaining these top components balances dimensionality reduction with information preservation.

## 5. Data Mining Techniques

As mentioned previously, this project used Logistic Regression, Decision Tree Algorithms, Naïve Bayes, MLP Classifier, and KNN as classification algorithms. Using the outputs of the top-performing base classifiers, an ensemble stacking classifier is constructed to generate the final predictions. In all cases, models are trained on the dataset's Class label using the 28 principal components derived from PCA as features. The dataset was divided into training and testing subsets using an 80 to 20 split. Stratified sampling was applied to ensure that both subsets preserved the original class distribution between "Trojan" and "Benign" traffic instances. This training to testing split was used for all classifiers.

### 5.1 *Logistic Regression*

Logistic Regression is a supervised learning algorithm used for binary classification tasks. It estimates the probability that a given input belongs to a particular class using a logistic function. The model assumes a linear relationship between the independent variables and the log-odds of the dependent variable, defined as:

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_x + ... + \beta_n x_n$$

In this project, $P(y=1)$ represents the probability of the positive class (trojan). The coefficients are learned through optimization. Using the Scikit Learn Logistic Regression function, the model was trained on the 28 principal components obtained from the PCA transformation. The logistic regression model applies the "saga" solver, which is suitable for large datasets and supports both $L_1$ and $L_2$ regularization. For this experiment, an $L_2$ penalty was selected to prevent overfitting by discouraging excessively large coefficient

weights. The regularization strength was controlled by setting the inverse parameter $C = 1.0$, which provides a balanced trade-off between model bias and variance. To ensure equitable learning across classes. The model was permitted up to 4000 optimization iterations to guarantee convergence.

5.2 *Decision Tree Algorithms*

Decision Tree classifiers are supervised learning algorithms used for classification. They recursively partition the feature space into subsets based on attribute values that maximize class separation. The algorithm selects a feature as a split threshold that yields the greatest information gain, measured by entropy:

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where $p_i$ represents the proportion of instances belonging to a subset defined by current node. The tree-growing process in Scikit-Learn continues until all leaves are pure or cannot be split further. Stopping criteria can be specified to control tree complexity and prevent overfitting. However, in this project, the depth was left unconstrained to allow the model to fully capture relationships within the PCA-reduced feature space.

5.3 *Naïve Bayes Classifier*

The Naïve Bayes classifier is a probabilistic model based on Bayes' Theorem:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X}$$

It assumes that features are conditionally independent given the class label. This assumption greatly simplifies computation. In this project, a Guassian Naïve Bayes variant was used and implemented via Scikit Learn. This version assumes that the continuous features follow a normal distribution within each class. For each feature $x_i$, the likelihood term is modeled as:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{(x_i-\mu)^2}{2\sigma^2}\big)$$

During training, the model estimates these class-specific means and variances directly from the training data. Prediction then involves computing the posterior probability for each class and assigning the class label with the highest probability.

5.4 *Random Forest Classifier*

Random Forest (RF) is an ensemble learning method that constructs a large number of decision trees on bootstrapped samples of the training set and aggregates their predictions. At each split, a random subset of features is considered, which de-correlates the trees and reduces variance relative to a single decision tree. In our model, we selected 300 trees with no depth limits.

5.5 *K-Nearest Neighbors*

K-Nearest Neighbors predicts a label for a point by examining the labels of the $k$ closest training samples in feature space. Closeness is measured by a distance metric:

$$d(x, x') = \big(\sum_{j=1}^{p} |x_j - x'_j|^2\big)^{1/2}$$

In this project, this algorithm was implemented as a supervised learning classifier to distinguish between "Trojan" and "Benign" network traffic instances. The classifier was configured with $k = 5$ neighbors and was set so that closer neighbors exert a stronger influence on classification than those farther away. During

prediction, each test sample's label was determined by a weighted vote among its five nearest neighbors from the training set.

### 5.6 *Ensemble Stacking*

The final classification technique used in this project was an ensemble stacking classifier. The outputs of multiple base classifiers were combined in this technique. Stacking integrates the strengths of individual models by training a secondary meta-learner that learns how to best combine their predictions. In this project, the outputs of the Decision Tree, Random Forest, and K-Nearest Neighbors (KNN) classifiers were selected as meta-features (had significantly better results), each representing a binary prediction for every instance in the dataset. The model outputs were weighted according to their individual accuracies obtained from prior evaluation. This ensured that models with higher predictive reliability had proportionally greater influence in the final stacked prediction. A Logistic Regression model served as the meta-learner.

## 6. Results

### 6.1 *Logistic Regression Results*

The logistic regression classifier achieved an accuracy of 53.61%. This model performed slightly better than a coin flip model would in predicting trojan and benign instances in our test set. The model had a higher tendency to predict instances as benign. The model correctly identifies a reasonable proportion of benign flows, but also produces a noticeable number of false positives. The model struggles to identify Trojan instances, detecting less than half of them. When the model did predict Trojan, it was more precise.

Table 2: Logistic Regression Classification Results

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| Benign | 0.5203 | 0.6545 | 0.5797 | 15617 |
| Trojan | 0.5613 | 0.4228 | 0.4823 | 16330 |
| **Accuracy** | | 0.5361 | | |
| **Macro Average** | 0.5408 | 0.5387 | 0.5310 | 31947 |
| **Weighted Average** | 0.5413 | 0.5361 | 0.5299 | 31947 |

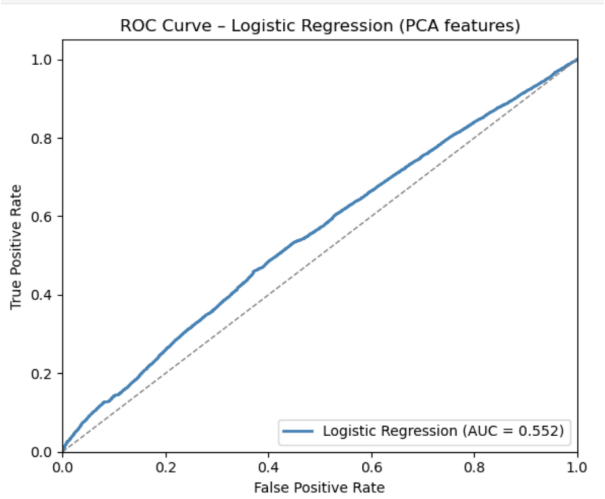


Figure 4: ROC Curve for Logistic Regression

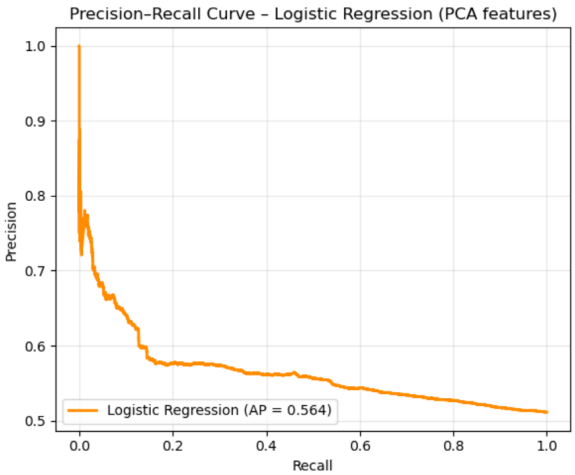


Figure 5: Precision–Recall Curve for Logistic Regression

The Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (Recall) against the False Positive Rate at varying decision thresholds. The diagonal dashed line represents a random classifier (AUC = 0.5). The Logistic Regression model achieved an Area Under the Curve (AUC) of 0.552, which indicates that its ability to discriminate between Trojan and Benign traffic is only marginally better than random guessing.

The Precision–Recall (PR) curve highlights the trade-off between precision and recall (the ability to find all positive instances). The model achieved an Average Precision (AP) of 0.564, consistent with the ROC-AUC result. Precision drops rapidly as recall increases, meaning that the model identifies more Trojans only at the cost of many false positives. This behavior is common in weak classifiers on imbalanced or noisy datasets, where confidence scores lack separation between classes.

### 6.2 *Decision Tree Classification Results*

The Decision Tree classifier achieved an overall accuracy of 64.90%. The model performed consistently across both classes, with similar precision, recall, and F1-scores for Benign and Trojan classes.

Table 3: Decision Tree Classification Results

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| Benign | 0.6407 | 0.6421 | 0.6414 | 15617 |
| Trojan | 0.6570 | 0.6557 | 0.6563 | 16330 |
| **Accuracy** | 0.6490 | | | |
| **Macro Average** | 0.6488 | 0.6489 | 0.6489 | 31947 |
| **Weighted Average** | 0.6490 | 0.6490 | 0.6490 | 31947 |

### 6.3 *Naïve Bayes Results*

The Naïve Bayes classifier achieved an overall accuracy of 51.24%, performing only marginally better than random guessing. The model's recall for benign instances (0.7976) was notably higher than for Trojan traffic (0.2396), indicating that it successfully identified most benign flows but failed to detect the majority of Trojan instances. This asymmetry reflects a strong bias toward predicting the benign class.

While the precision for Trojan predictions (0.5532) was slightly higher than for benign (0.5008), the model's F1-score for Trojans (0.3344) reveals poor balance between precision and recall. This suggests that the model often misclassifies Trojans as benign.

Table 4: Naïve Bayes Classification Results

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| Benign | 0.5008 | 0.7976 | 0.6153 | 15617 |
| Trojan | 0.5532 | 0.2396 | 0.3344 | 16330 |
| **Accuracy** | 0.5124 | | | |
| **Macro Average** | 0.5270 | 0.5186 | 0.4748 | 31947 |
| **Weighted Average** | 0.5276 | 0.5124 | 0.4717 | 31947 |

Overall, Naïve Bayes struggled in this context because it assumes that features are conditionally independent. This condition is not true in network traffic data. The relatively low accuracy and unbalanced recall imply that this simple probabilistic model is insufficient to capture the nonlinear dependencies among the PCA-transformed features.

6.4 *Random Forest Results*

The Random Forest classifier achieved an overall accuracy of 68.76% and an ROC-AUC score of 0.7669. The model demonstrates balanced performance across both classes, with similar precision and recall for Benign (0.6769 / 0.6907) and Trojan (0.6983 / 0.6848) instances. This indicates that Random Forest was effective in identifying both types of traffic without a strong bias toward one class.

Table 5: Random Forest Classification Results

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| Benign | 0.6769 | 0.6907 | 0.6837 | 15617 |
| Trojan | 0.6983 | 0.6848 | 0.6915 | 16330 |
| **Accuracy** | 0.6876 | | | |
| **Macro Average** | 0.6876 | 0.6877 | 0.6876 | 31947 |
| **Weighted Average** | 0.6879 | 0.6876 | 0.6877 | 31947 |
| **ROC AUC** | 0.7669 | | | |

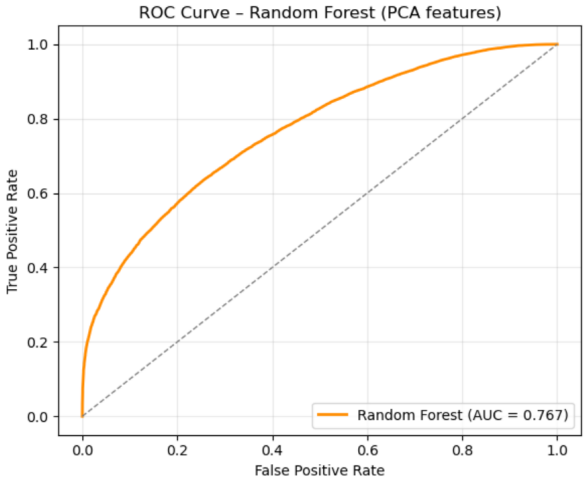


Figure 6: ROC Curve for Random Forests

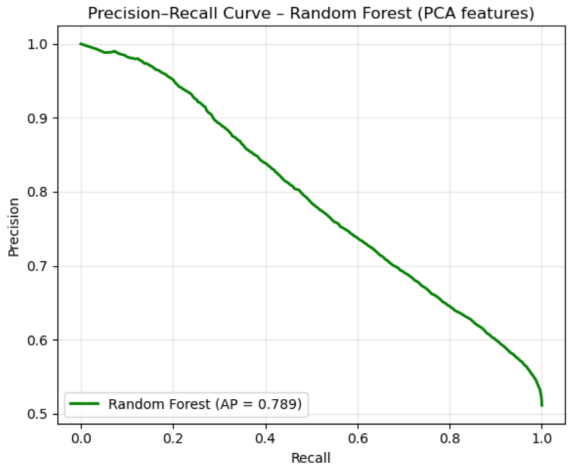


Figure 7: Precision–Recall Curve for Random Forest

The ROC curve for the Random Forest model shows a consistent rise above the diagonal reference line, achieving an Area Under the Curve (AUC) of 0.767. This indicates strong separability between Trojan and Benign classes. The model maintains a good balance between true positive rate (sensitivity) and false positive rate across various decision thresholds. The Precision–Recall (PR) curve further highlights the model's effective performance, with an Average Precision (AP) of 0.789. Precision remains high across a wide range of recall values, meaning that the Random Forest can detect many Trojan instances without significantly increasing false positives.

6.5 *K-Nearest Neighbors Classification Results*

The K-Nearest Neighbors (KNN) classifier achieved an overall accuracy of 67.17%. he model shows strong balance across both classes, with similar precision, recall, and F1-scores for Benign and Trojan instances. KNN's performance suggests that samples of similar network flow characteristics tend to cluster naturally in the reduced-dimensional space—allowing KNN to leverage local structure for classification.

Table 6: K-Nearest Neighbors (KNN) Classification Results

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Benign | 0.6579 | 0.6842 | 0.6708 | 15617 |
| Trojan | 0.6860 | 0.6597 | 0.6726 | 16330 |
| **Accuracy** | 0.6717 | | | |
| **Macro Average** | 0.6719 | 0.6719 | 0.6717 | 31947 |
| **Weighted Average** | 0.6722 | 0.6717 | 0.6717 | 31947 |

### 6.6 *Ensemble Stacking Results*

The Ensemble Stacking Classifier achieved an overall accuracy of 68.26%, performing competitively with the Random Forest model (68.76%) and surpassing all other individual base classifiers. The model demonstrates balanced precision and recall across both Benign and Trojan classes, with nearly identical F1-scores ($\approx 0.68$) for each. This indicates that the ensemble effectively integrates the strengths of the base models to produce stable, well-calibrated predictions.

Table 7: Ensemble Stacking Classifier Results (Meta-Model on Base Predictions)

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Benign | 0.6706 | 0.6894 | 0.6799 | 15617 |
| Trojan | 0.6948 | 0.6761 | 0.6853 | 16330 |
| **Accuracy** | 0.6826 | | | |
| **Macro Average** | 0.6827 | 0.6828 | 0.6826 | 31947 |
| **Weighted Average** | 0.6830 | 0.6826 | 0.6827 | 31947 |

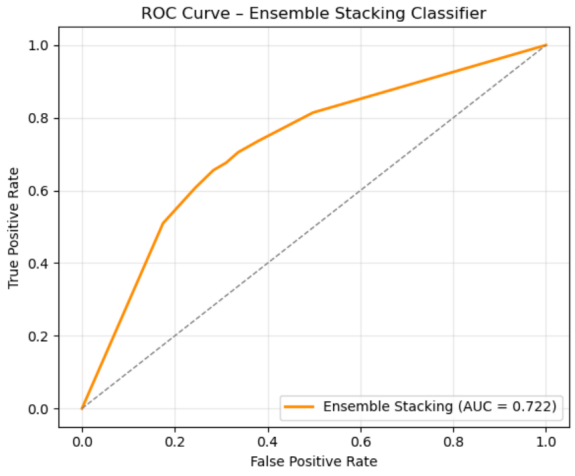


Figure 8: ROC Curve for Ensemble Stacking

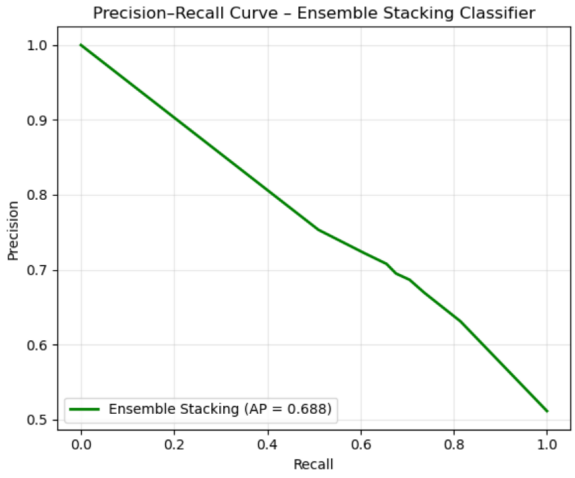


Figure 9: Precision–Recall Curve for Ensemble Stacking

## 7. Conclusion

Simpler models such as Logistic Regression and Naïve Bayes performed below expectations, with accuracies of 53.6% and 51.2%, respectively. Their limited capacity to model nonlinear relationships in the PCA-reduced feature space resulted in weaker discrimination between trojan and benign flows. These models primarily captured global trends, but failed to generalize well across the diverse network traffic patterns present in the dataset.

Tree-based and distance-based learners achieved stronger results. The Decision Tree reached an accuracy of 64.9%, while the K-Nearest Neighbors and Random Forest classifiers achieved 67.2% and 68.8%, respectively.

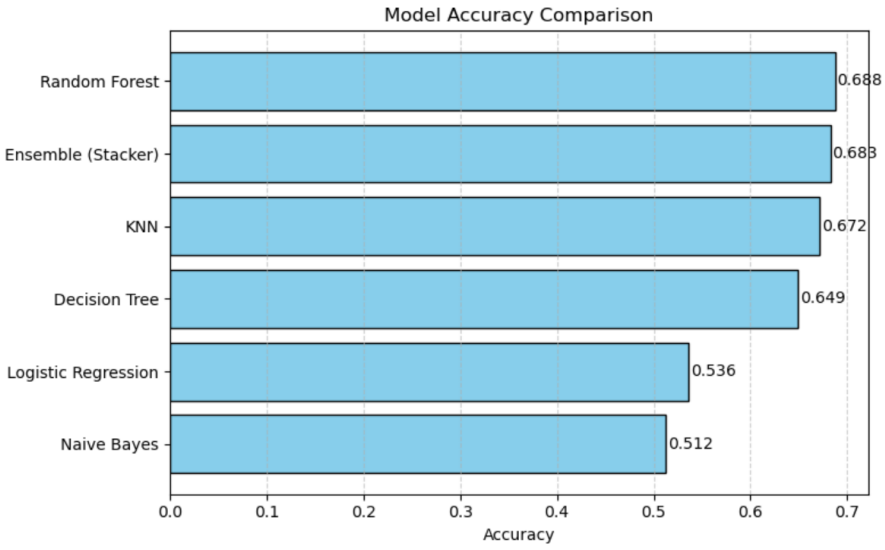


Figure 10: Accuracy among all predictive models

The Ensemble Stacking Classifier demonstrated competitive and balanced performance across all key metrics. While its AUC was slightly lower than that of Random Forest, the stacking model offered a more stable balance between precision and recall, reducing variance across predictions. By combining outputs from Decision Tree, Random Forest, and KNN through a logistic meta-model, the ensemble captured complementary decision boundaries from each base classifier. The hybridization led to smoother, more generalized decision surfaces—making the stacking approach the most reliable and consistent overall classifier in the experiment.

Future work will focus on improving the overall classification performance through more extensive hyper-parameter optimization and meta-learner tuning. Fine-tuning the ensemble's base classifiers—particularly KNN's neighborhood size, Random Forest's tree depth, and the stacking weights—could yield more stable and accurate predictions. Additionally, incorporating genetic algorithms could benefit both feature selection and model parameter optimization. By evolving combinations of hyper-parameters and feature subsets based on fitness metrics such as AUC or F1-score, the model could achieve improved generalization and reduced overfitting. Integrating such evolutionary optimization methods in future iterations of this project could lead to more adaptive, high-precision detection of trojan network traffic patterns.

## References


[1] Wang, Zihao, Kar Wai Fok, and Vrizlynn L. L. Thing. "Machine Learning for Encrypted Malicious Traffic Detection: Approaches, Datasets and Comparative Study." Computers & Security, vol. 113, 2022, 102542. *ScienceDirect*, https://doi.org/10.1016/j.cose.2021.102542

[2] Azab, Ahmad, Mahmoud Khasawneh, Saed Alrabaee, Kim-Kwang Raymond Choo, and Maysa Sarsour. "Network Traffic Classification: Techniques, Datasets, and Challenges." *Digital Communications and Networks*, vol. 10, no. 3, 2024, pp. 676-692. ScienceDirect, https://doi.org/10.1016/j.dcan.2022.09.009.

[3] Jia, Zijian, Yepeng Yao, Qiuyun Wang, Xuren Wang, Baoxu Liu, and Zhengwei Jiang. "Trojan Traffic Detection Based on Meta-learning." *Lecture Notes in Computer Science – ICCS 2021: Computational Science*, Springer, 2021, pp. 167-180. SpringerLink, https://doi.org/10.1007/978-3-030-77964-1$_1$4.

[4] Biswas, Abhishek. "Anomaly Detection with BIRCH." *AI in Plain English*, Medium, 7 Sept. 2023. Available at: https://ai.plainenglish.io/anomaly-detection-with-birch-7f0b5f35ed16.

[5] Pedregosa, Fabian, et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2825–2830.


*Submitted by Matthew Poncini on 23 October 2025.*