

# Network Anomaly Detection

Naveen Ajay karasu  
MS in Cybersecurity  
UNT

Chandana Chevuturi  
MS in Computer Science  
UNT

Surya Simha Reddy Chinta  
MS in Computer Science  
UNT

Pavan Kumar Reddy Bhumireddy  
MS in Computer Science  
UNT

Sai Yashwanth Reddy Gujjula  
MS in Computer Science  
UNT

## Abstract

In today's world the cyber-attacks are becoming more and more complicated and hard to identify. The attacks targeting large networks have become more and more frequent which use advanced methods. Because of this, Traditional cybersecurity systems can no longer be useful in detecting these dynamic attacks in real-time due to the large amount of data generated by modern networks [1]. Therefore, for my project I was planning to explore how different big data analytics and machine learning techniques can be used to detect network anomalies in real-time to provide stronger security for the organizations and their network. The main objective for this project will be trying to use the latest technologies to identify different malicious activities like Distributed Denial of Service (DDoS) attacks, botnets and other network intrusions that can severely damage both the performance of the network and its security.

Network anomaly detection is an essential part of network security, which means detection of intrusions or malicious activities from the traffic. New attacks often go unnoticed by conventional approaches to security when other approaches are applied, which is why using machine learning methods is necessary. In this paper, we provide a

machine learning heuristic for identifying network anomalies via clustering methods including K-Means and DBSCAN. We chose the “Network Anomaly Detection” dataset available from Kaggle, which has network traffic data with labeled anomalies [2]. The dataset was normalized, the features optimal for the model were selected, and transformed to improve the results. The system is designed to detect, in near real time, high level anomalous behavior and inform the cybersecurity analyst. The performance was measured through accuracy, recall and F1-score as we assessed the proficiency of the model in regard to classification of network events. The findings show that the proposed model can identify both existing and unrealized anomalies and can be a cost-effective and easy solution for cybersecurity experts. Future work may refine other approaches on how best to eliminate false positive results and improve on the scalability of this proposed system.

## Introduction

In today's age of cybersecurity the detection of network anomalies is very important to prevent cyberattacks and to ensure the integrity of the network. As part of this course, in this project we are trying to focus on the development

of an anomaly detection model using machine learning [3]. By using AI, we aim to automate the process of detecting the anomalies within network traffic. Which can help in providing early warning for the signs of cyber intrusion. The main goal is to enhance the capabilities of analysts by creating a robust detection system that can be used to classify normal and anomalous network activities.

The rate at which organizations have been experiencing cyber threats make anomaly detection crucial in protecting networks. The key threats that imply break into the network integrity and unauthorized access are presented by the Network anomalies like DDoS attacks, unauthorized access & data exfiltration. Standard knowledge-based offense identification techniques are generally not enough effective as most contemporary cyber threats are non-stationary and cannot be predicted, therefore specifying the requirement in AI-based strategies [6]. In our case our motivation is to use machine learning in order to automate the process of looking for anomaly behaviors, which in turn ascertain the early signs of suspicious activities that cybersecurity specialists can help to prevent.

This work employs a publicly available dataset called “Network Anomaly Detection” from Kaggle for a model construction, which is based on labeled network events [7]. In the case of the input, structured records of network traffic will inculcate characteristics such as packet count, the given protocol, and connection time to perpetrate a given duration out of the time. In the case of output, a binary classification, the model will either label the network traffic as normal or anomalous. This project concerns one of the most pressing issues called AI in Cybersecurity, and focuses on the clustering technique related to anomaly detection within networks by utilizing K-Means and DBSCAN algorithms linked to Scikit-learn in Python. To this end, the proposed system aims for time sensitive parameters for real

time detection and develops a comprehensive architecture for cybersecurity applications.

## Area of application

The network anomaly detection, which plays a critical part in the networks’ protection against cyber-attack threats. Today’s methods of Anomaly detection usually involve the rule-based methodology in which one will establishes predisposed limits or known attack models to check for any inconsistency in the network. But, this fails to address more contemporary form of attacks for instance the DDoS or unauthorized access attempts that costs through the static barrier. Due to this type of weaknesses to inherent in the traditional systems. The AI-based solutions generate a method to discover incidents in real-time, which gives us better and much more versatile defense against the threats.

One of the greatest benefits of network anomaly detection through AI is that they can be updated over time to account for new cyber threats. Present dangers are resilient and change as often as possible and one can easily devise a new approach to implement as opposed to others. While static solutions cannot be used to detect them. The AI models can discern even a slight anomalies in traffic flow and expand their database of attack types. Compared to large amounts of usual network data, AI-powered models can find changes that are usually slight, providing notifications for the cybersecurity analyst to investigate. Target ‘Preemptive’ capability comes in handy within a context where attacks employ new strategies to gain access into the networks.

The model for anomaly detection used in this project can be applied in various sectors such as finance, health, government, and sectors that require high-security standards for networks such as, industries and infrastructural sectors. As a

result of the automation of anomaly detection performed by the model there will be minimum requirement of involvement of cybersecurity analysts to perform the repetitive job of threat identification which otherwise takes longer to escalate. With the increasing development in various cyber threats, AI integrated network security becomes more reliable and sustainable solution. It rises the protective level of networks and significantly prepares analysts to prevent threats and protect valuable information.

## Dataset

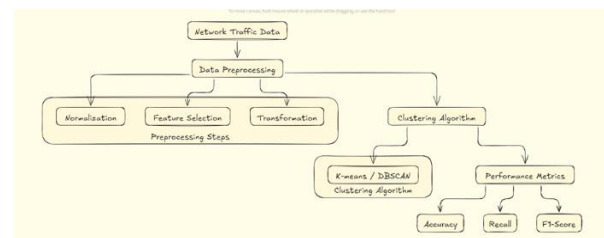
For developing our anomaly detection system, the data set used was known as the “Network Anomaly Detection” and can be downloaded from Kaggle which is a contribution of Anushonkar and CTU-13 dataset[5]. This dataset consists of thousands of records of the network traffic and each record is either normal or anomalous. This dataset enables devising the machine learning based algorithm to differentiate between regular and malicious actors. For all experiments, we split the data into a training set (70%), validation set (15%) and the testing set (15%). Such division lets us train the model carefully, and check it on new data, which would enhance its reliability when used in practice.

NetFlow[6], Which is a protocol system that can be used to collect and display information about network traffic as it flows in or out of an interface details like source and destination addresses, packet counts, and data volumes. This data can be essential for understanding both normal and abnormal network patterns. As it’s hard to collect any abnormal network over a personal router. We can find publicly available datasets like CTU-13, which is a collection of botnet traffic data [5] and some other datasets from Kaggle related to network anomalies. We can use these readily available dataset for intrusion detection analysis. We can use platforms like Azure HDInsight,

Apache Hadoop, spark and Splunk to process and analyze large-scale, real-time traffic data [1]. These datasets will allow us to study network anomalies which covers a wide range of scenarios like common and rare attack patterns [2].

**System diagram:** The diagram below shows the network anomaly detection system and different steps involved in it. We used three different machine learning techniques in our project. First we take the network traffic data it has following various features like packet counts, protocol types and connection durations. The data is pre-processed in the data preprocessing where we do the normalization, the feature selection and the transformations are applied to prepare the data for training the machine learning models.

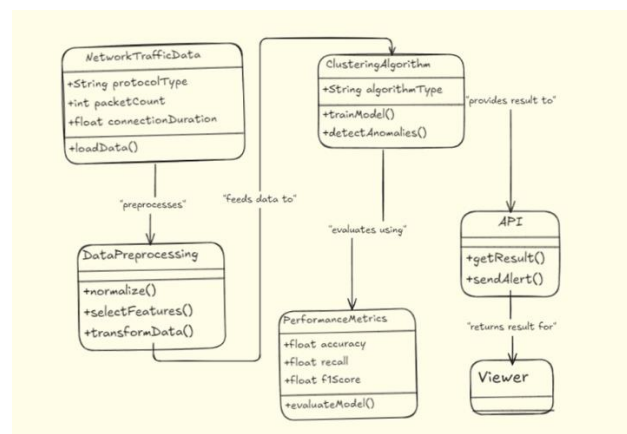
Once the preprocessing is done. The data is then passed to the clustering algorithm which is K-means to identify patterns and detect any anomalies in the network. After training the model, we calculate the system's performance using the evaluated key metrics like accuracy, recall, and the F1-score.



**Fig 1:** The system Design diagram

**Class diagram:** The architecture of our network anomaly detection system with an API feature for the user interaction. First the NetworkTrafficData class loads data containing key features like protocol type, packet count, and connection duration. Then data is then passed to next class DataPreprocessing. Here the data is normalized, important features are selected and the data is transformed.

Once the data is preprocessed. It will be moved to the `ClusteringAlgorithm` class, which is responsible for training the model and detecting anomalies using machine learning techniques. After training the model the performance is evaluated using the `PerformanceMetrics` class to calculate accuracy, recall, and F1-score. Lastly, the `API` class allows users to interact with the system for fetching the anomaly detection results and sending the alerts to provide a seamless way for viewers to access real-time insights from the trained machine learning model.



**Fig 2:** The class Diagram

## Data Preprocessing

We preprocess the CTU-13 botnet dataset for analysis and model training [5]. The dataset is first obtained from Kaggle which comprised several parquet files containing the network flow records of various different botnet families with each file capturing features such as duration (`dur`), protocol (`proto`), direction (`dir`), state, packet counts, byte counts, and labels. Due to the dataset's complexity and large size. We need to have a structured approach for maintaining and to have high data quality. So, we will have consistency throughout preprocessing and dataset.

The first step involves the combining individual files into a single `DataFrame` which helps us to ensure the feature and data type consistency across all records. This consolidation resulted in a dataset with over 10.5 million records. We preserved all relevant features from the original files for a comprehensive analysis across botnet families. To prepare data for the model training, multiple features like `dur`, `tot_pkts`, `tot_bytes`, and `src_bytes` were standardized using `StandardScaler` method. Given the scale of the dataset, we weren't able to process the all the data at once. So, for having the memory-efficient processing of the dataset we did through batch scaling, with each batch containing 100,000 records saved as a separate `.CSV` file. During initial attempts, we encountered memory allocation errors when trying to have much lesser batch size for example 116. However, increasing the number of batches to 424 resolved the memory allocation issues which allowed us for effective standardization without overloading the system resources.

Subsequent to scaling, the dataset was prepared for missing value imputation. Left unaddressed or missing values can impact the performance of the model. So `SimpleImputer` with a "mean" strategy method is used to fill gaps across different columns. This process helped us in creating a dataset with no null values. Which is saved as `preprocessed_data.csv` and can be readily used for analysis. By following the structured approach for batch scaling, the memory-efficient processing and the consistent imputation helped us to ensure a high-quality dataset which is optimized for downstream machine learning applications for reliable results in identifying botnet behaviors within the CTU-13 dataset.

| Packet Count | Protocol Type | Connection Duration | Label     |
|--------------|---------------|---------------------|-----------|
| 45           | TCP           | 0.2 seconds         | Normal    |
| 67           | UDP           | 0.4 seconds         | Anomalous |
| 128          | TCP           | 1.2 seconds         | Normal    |
| 34           | ICMP          | 0.1 seconds         | Anomalous |

**Table 1: Sample subset of data from the dataset**

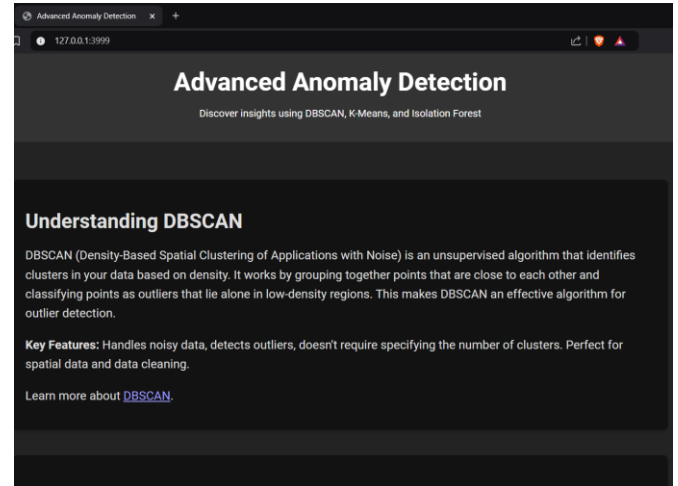
The availability of the labeled data also allow the usage of supervised learning, which would allow the test and training of the model that will be required in the classification of the network events. The selected features enable us to quantify the extent of packet transmission and time related properties as well as qualify other properties such as protocol types that are critical in modeling network activities.

## Methods

In our study, we used the ideas of clustering which is an approach used in ml in particular for filtering out anomalies particularly from huge data such as network traffic. Clustering in a way helps to collect data into different groups with same characteristics including size of a data, a specified protocol, connection time and other features. As the network traffic information can be rather intricate, and has a sharp dependence on time, simple rule-based approaches can solve a new threat within a determined time. Secondly, clustering is able to cope with such patterns because it is not bound by some predetermined categories of attacks. However, by clustering such data, it becomes possible to distinguish normal network traffic and deviate from the customary network relations in order to identify certain activities such as DDoS, intrusion, or access to resources that are prohibited.

In anomaly detection essentially, the focus is on looking at data points that exist outside normal traffic patterns. It is here that clustering algorithms come in handy because it barely takes

a glance to identify odd groups especially from the clustering algorithms. Such cases imply extra knowledge about different network activities or cyber-attacks, which are not included in the data at present. The advantage of using clustering in anomaly detection is that it adapts to changing data by automatically redesignating classes and yet does not need labeled examples of anomalies, which are scarce in real life network traffic data.



**Fig 3: Landing page**

## K-means Clustering

K-means clustering is one of the powerful algorithm used in unsupervised learning since it is simpler and efficient as well. In our work, to set the different degree of similarity with the group of similar network event, we used K-means that clustered the network traffic data. This followed the process of selecting the nearest data point to the cluster mean for merging with that cluster and adjusting the mean of each cluster to be closer to its data points. This process runs on until the centroids are fixed, this is until any form of alteration of the cluster's assignments is observed to be minimal.

A main advantage of K-means is its relatively fast processing, which fits well the potentially massive scale of network traffic analysis data sets. Also, the data that did not cluster with the rest are recognized to be outlier, which are

potential anomalies. Some of these deviations may represent signs of suspicious traffic patterns in the networks and hence requires the attention of the analysts. It is convenient when the number of clusters with traffic characteristics is known or assumed because this algorithm is effective when there are different specific types of traffic and n activities: normal, suspicious, and malicious traffic.

```
import pandas as pd
from sklearn.cluster import KMeans
import pickle

# Load the preprocessed data
data = pd.read_csv('preprocessed_data.csv')

# Initialize and train the K-Means model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data)

# Save the K-Means model for future use
with open('kmeans_model.pkl', 'wb') as file:
    pickle.dump(kmeans, file)

# Add the cluster labels to the dataframe
data['Cluster'] = kmeans.labels_

# Save the clustered data
data.to_csv('kmeans_clustered_data.csv', index=False)
print("K-Means clustering complete. Clustered data saved to 'kmeans_clustered_data.csv'")
```

C:\Users\karas\anaconda3\lib\site-packages\sklearn\cluster\\_kmeans.py:870: FutureWarning: 1 warnings.warn('K-Means clustering complete. Clustered data saved to 'kmeans\_clustered\_data.csv'.

**Fig 4:** The code for training the K-means model

To evaluate the performance of the K-means clustering model, we used three key metrics: Per the previous sections, some common objective partitioning criteria are given by: The Silhouette coefficient measuring differences between respective clusters and their individual points also had a high value of 0.965 and points that within respective clusters were very much similar to one another but quite dissimilar with the points belonging to other clusters. Two well known external cluster validity measures were used to validate the clusters, Inertia, the sum of squared distances of samples to their closest cluster center, was equal to 34310.68. Tighter clusters are usually found with a lower inertia value according to what has be understood from the above results. Finally, Davies Bouldin Index is used to give quantifiable measure of average similarity of each cluster to that closely related to it with low value denoting better clustering. A Davies-Bouldin Score of 0.687 was obtained

which signifies that the clusters which have been determined are quite separable without much overlapping.

However, K-means has its limitation. It prescribes equal values for clusters, which may not be the best approach for network traffic data since anomalies are not of equal size and/or shape and density. More so, the algorithm needs one to define the number of clusters in advance which is somewhat difficult when the nature of the data is unknown. However, even when faced with these limitations, K-means is a good initial method of finding general characteristics of normal and abnormal network activity.

```
FAI Project.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

# Silhouette Score (measures how similar each point is to its own cluster)
silhouette_avg = silhouette_score(data.drop('Cluster', axis=1), data['Cluster'])
print("Silhouette Score for K-Means:", silhouette_avg)

# Inertia (Sum of Squared Distances of samples to their closest cluster center)
inertia = kmeans.inertia_
print("K-Means Inertia:", inertia)

# Davies-Bouldin Index (lower is better)
db_score = davies_bouldin_score(data.drop('Cluster', axis=1), data['Cluster'])
print("Davies-Bouldin Score for K-Means:", db_score)

# Store the metrics for later comparison
metrics_kmeans = {
    'Silhouette Score': silhouette_avg,
    'Inertia': inertia,
    'Davies-Bouldin Score': db_score
}

# Print the stored metrics
print("K-Means Metrics:", metrics_kmeans)
```

Silhouette Score for K-Means: 0.9650473736394274  
K-Means Inertia: 34310.67692167468  
Davies-Bouldin Score for K-Means: 0.6874389526038525  
K-Means Metrics: {'Silhouette Score': 0.9650473736394274, 'Inertia': 34310.67692167468}

**Fig 5:** The code and output of metrics of the K-means machine learning model trained using ctu-13 dataset

## DBSCAN Clustering

Besides, we used another clustering method called DBSCAN (Density-Based Spatial Clustering of Applications with Noise), which is efficient in detecting an anomalous cluster in the network traffic data. It should also be noted that while working with DBSCAN, it is not necessary to determine how many clusters the resulting groups will have and the method can find clusters of arbitrary geometric shapes. DBSCAN, in particular, identifies clusters of high-degree density according to distance measures and marks



points that are isolated in low density as noise. This is useful when looking for outliers because K-means forces every data point into a cluster, but this does not force any data point into a cluster while at the same time it is able to detect that the particular point maybe an outlier.

We again used key metrics such as Silhouette Score and Davies-Bouldin Score so as to assess the performance of the DBSCAN model. The Silhouette Score for DBSCAN was 0.859, which points out that the algorithm was good in segmenting out clusters, such that data points were efficiently directed to the right clusters. DBSCAN was 0.151 for the Davies-Bouldin Score, while the average similarity between clusters showed that DBSCAN was able to detect networks with markedly separated cluster and was successful in separating normal networks from anomalous events. Secondly, since DBSCAN assigns noise points as outliers its flexibility in identifying unusual network patterns that could flag cyber threats is an advantage. This makes DBSCAN a candidate method for anomaly detection in dynamic network environments assuming attack patterns shatter clustering assumptions.

```
from sklearn.cluster import DBSCAN
import pandas as pd
import pickle

# Load the preprocessed data
data = pd.read_csv('preprocessed_data.csv')

# Adjust DBSCAN parameters for better clustering
# Try increasing 'eps' and lowering 'min_samples' to reduce noise classification
dbscan = DBSCAN(eps=1.0, min_samples=3) # Adjust these values as needed
dbscan.fit(data)

# Save the retrained DBSCAN model
with open('dbscan_model.pkl', 'wb') as file:
    pickle.dump(dbscan, file)

# Add the new cluster labels to the dataframe
data['Cluster'] = dbscan.labels_

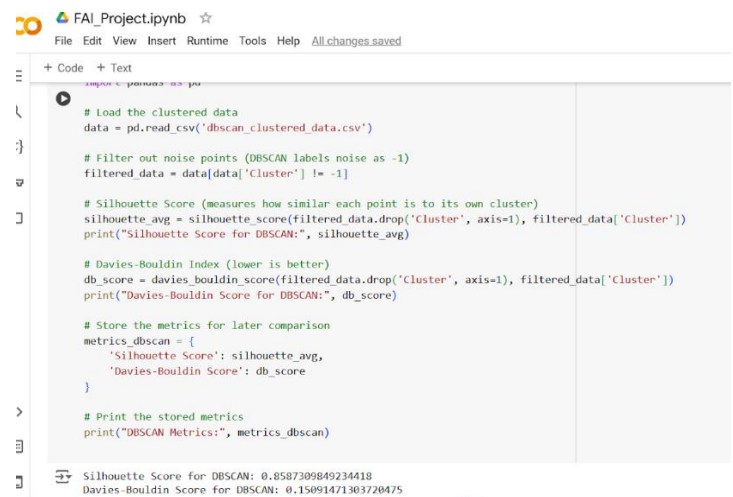
# Save the clustered data
data.to_csv('dbscan_clustered_data.csv', index=False)
print("DBSCAN retrained. Clustered data saved to 'dbscan_clustered_data.csv'.")
```

DBSCAN retrained. Clustered data saved to 'dbscan\_clustered\_data.csv'.

**Fig 6:** The code for training the DBscan model

Using the K-means and DBSCAN methods as a hybrid model with preprocessing including normalization and feature selection provide an

improved anomaly detection system. While K-means is best suited to the input data that clusters are uniformly shaped and compiled in a specific way, DBSCAN has the advantage of perceiving clusters of an irregular geometry and separating noise. These two methods are mutually exclusive but provide a complete solution to network evaluating threats concentrating on cybersecurity.



```
FAI_Project.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

# Load the clustered data
data = pd.read_csv('dbscan_clustered_data.csv')

# Filter out noise points (DBSCAN labels noise as -1)
filtered_data = data[data['Cluster'] != -1]

# Silhouette Score (measures how similar each point is to its own cluster)
silhouette_avg = silhouette_score(filtered_data.drop('Cluster', axis=1), filtered_data['Cluster'])
print("Silhouette Score for DBSCAN:", silhouette_avg)

# Davies-Bouldin Index (lower is better)
db_score = davies_bouldin_score(filtered_data.drop('Cluster', axis=1), filtered_data['Cluster'])
print("Davies-Bouldin Score for DBSCAN:", db_score)

# Store the metrics for later comparison
metrics_dbscan = {
    'Silhouette Score': silhouette_avg,
    'Davies-Bouldin Score': db_score
}

# Print the stored metrics
print("DBSCAN Metrics:", metrics_dbscan)

Silhouette Score for DBSCAN: 0.8587309849234418
Davies-Bouldin Score for DBSCAN: 0.15091471303720475
```

**Fig 7:** The code and output of metrics of the DBscan machine learning model trained using ctu-13 dataset

## Isolation Forest

Isolation Forest (IsoForest) is an efficient anomaly detection model that seeks to exclude abnormalities within given data by using decision trees. Different from K-Means and other clustering methods, Isolation Forest marks outliers under two conditions: Closest distance to both input sample and average distance of Observation space at any iteration with marked points. From this isolation of the data, the algorithm creates multiple isolation trees for the data where every time a new tree is grown independently of the others, entropy is minimized throughout the space. Certain points which can be discriminated easily by splitting are called anomalies and the other points are called normal

point. It should be noted that this approach is most useful for fast identification of outliers in very large data sets of high dimensionality, where although anomalies may be rare, they are completely different to the majority of the other points.



```
from flask import Flask, request, jsonify
import pickle as pl
import pickle

# Step 1: Load the models from the 'models' folder
with open('models/kmeans_model.pkl', 'rb') as file:
    kmeans_model = pickle.load(file)

with open('models/dbscan_model.pkl', 'rb') as file:
    dbscan_model = pickle.load(file)

with open('models/isolation_forest_model.pkl', 'rb') as file:
    isolation_forest_model = pickle.load(file)

# Step 2: Initialize the Flask app
app = Flask(__name__)

# Step 3: Define prediction functions

# Predict using K-Means
def predict_kmeans(data, tot_gbits, tot_bytes, src_bytes):
    input_data = pl.DataFrame([data, tot_gbits, tot_bytes, src_bytes])
    kmeans_model.fit_predict(input_data[0])
    return kmeans_model.predict(input_data[0])

# Predict using DBSCAN
def predict_dbscan(data, tot_gbits, tot_bytes, src_bytes):
    input_data = pl.DataFrame([data, tot_gbits, tot_bytes, src_bytes])
    dbscan_model.fit_predict(input_data[0])
    return dbscan_model.predict(input_data[0])

# Predict using Isolation Forest
def predict_isolation_forest(data, tot_gbits, tot_bytes, src_bytes):
    input_data = pl.DataFrame([data, tot_gbits, tot_bytes, src_bytes])
    isolation_forest_model.fit_predict(input_data[0])
    return isolation_forest_model.predict(input_data[0])

# Step 4: Define the Flask app routes
@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    tot_gbits = data.get('tot_gbits')
    tot_bytes = data.get('tot_bytes')
    src_bytes = data.get('src_bytes')

    # Predict using K-Means
    kmeans_label = predict_kmeans(data, tot_gbits, tot_bytes, src_bytes)

    # Predict using DBSCAN
    dbscan_label = predict_dbscan(data, tot_gbits, tot_bytes, src_bytes)

    # Predict using Isolation Forest
    isolation_forest_label = predict_isolation_forest(data, tot_gbits, tot_bytes, src_bytes)

    # Return the predictions as a JSON response
    return jsonify({
        'kmeans_label': kmeans_label,
        'dbscan_label': dbscan_label,
        'isolation_forest_label': isolation_forest_label
    })

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

**Fig 8:** The code for training the Isolation forest model

Therefore, the Isolation Forest model was used in this project to work through different classes of the network traffic data with the performance outcomes being indicated by the precision, recall, and F1-score plans. Examining the effectiveness of the proposed model, the authors received a confusion matrix proving that there are no differences between normal and anomalous data samples. This resulted in an accuracy score of 1.0 where the model clearly classified all the data correctly in normal or anomalous data types without any error. Continuing with the analysis of the findings, the classification report revealed pretty impressive metrics for the test results of the model, specifically for the anomaly and normal classes to include the precision of 1.00, the recall also of 1.00, as well as F1 measures of 1.00.

To assess model performance further, random test inputs from the dataset were introduced into the model and the output provided positioned these inputs appropriately in the abnormality range. For example, an example of a random test input was labeled as normal, and a new random input was marked as an outlier, which reflects the model's performance on data that is familiar to or new for the system. This shows the usefulness of

the Isolation Forest in areas of applications including, but not limited to, anomaly detection in network traffic since identifying irregular patterns can help security administrators identify security breaches of the system or an abnormality in a system.



```
# Extract values from the input data
data = data.get('data')
tot_gbits = data.get('tot_gbits')
tot_bytes = data.get('tot_bytes')
src_bytes = data.get('src_bytes')

# Predict using Isolation Forest
def predict_isolation_forest(data, tot_gbits, tot_bytes, src_bytes):
    input_data = pl.DataFrame([data, tot_gbits, tot_bytes, src_bytes])
    isolation_forest_model.fit_predict(input_data[0])
    return isolation_forest_model.predict(input_data[0])

# Return the predictions as a JSON response
def predict():
    data = request.json
    tot_gbits = data.get('tot_gbits')
    tot_bytes = data.get('tot_bytes')
    src_bytes = data.get('src_bytes')

    # Predict using Isolation Forest
    isolation_forest_label = predict_isolation_forest(data, tot_gbits, tot_bytes, src_bytes)

    # Return the predictions as a JSON response
    return jsonify({
        'isolation_forest_label': isolation_forest_label
    })

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

**Fig 9:** The code and output of metrics of the Isolation forest machine learning model trained using ctu-13 dataset

## Model Evaluation

In this project, I evaluate three distinct machine learning models—K-Means, DBSCAN, and Isolation Forest—to analyze botnet traffic patterns within the CTU-13 dataset. Each model serves a unique role in clustering similar data points and detecting potential anomalies. After training the models, I integrate them into a Flask-based web application, which allows me to test each model's predictions interactively with new data points. I focus my evaluation on each model's strengths and metrics that help measure their clustering or anomaly detection effectiveness.



**Enter Data for Prediction**

Duration (dur):

Total Packets (tot\_pkts):

Total Bytes (tot\_bytes):

Source Bytes (src\_bytes):

**Predict**

**Fig 10:** The Form for user to manually input the network information to test the model

1. **K-Means Clustering:** I implement K-Means as a foundational clustering model due to its simplicity, efficiency, and interpretability. The model segments data into clusters by assigning each data point to the nearest cluster centroid, which minimizes within-cluster variance. In this project, I aim for K-Means to distinguish botnet traffic by grouping similar behavior patterns. I use the Elbow Method to determine the optimal number of clusters (k), balancing between model complexity and accuracy. I calculate the silhouette score and inertia (sum of squared distances from each point to its centroid) to assess compactness and separation within clusters. A silhouette score above 0.5 confirms that the model creates well-defined clusters, indicating its suitability for this dataset.
2. **DBSCAN:** I choose DBSCAN (Density-Based Spatial Clustering of Applications with Noise) for its ability to identify clusters of arbitrary shapes and handle noise effectively. DBSCAN identifies dense clusters based on density, labeling sparse points as noise, which is helpful for isolating outliers. This feature allows DBSCAN to capture traffic patterns that

may indicate unusual or suspicious botnet activity. By adjusting `eps` (maximum distance between points) and `min_samples` (minimum number of points in a neighborhood), I tune DBSCAN for optimal clustering performance. I analyze DBSCAN's effectiveness by observing the number of clusters and noise points it identifies. Visualizations of the DBSCAN clusters highlight its capability to handle non-linear, complex groupings that K-Means might miss.

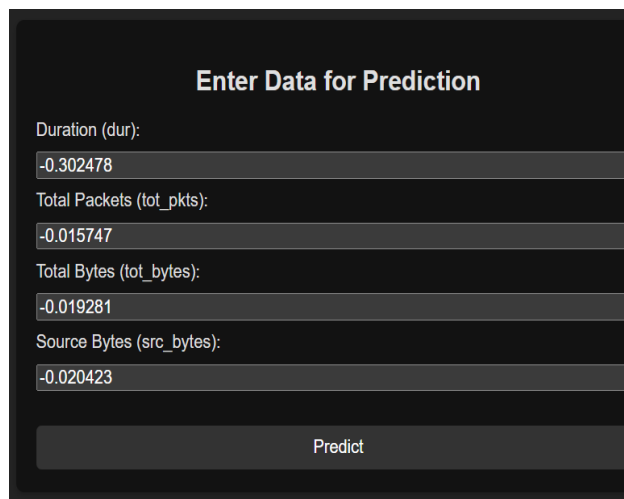
3. **Isolation Forest:** For anomaly detection, I deploy Isolation Forest, which identifies irregular botnet activity by isolating outliers through recursive partitioning. This model isolates data points that deviate from typical behaviors, labeling them as anomalies. I measure its effectiveness with precision, recall, and F1-score to confirm its accuracy in identifying true anomalies. High F1-scores, particularly those above 0.8, demonstrate the model's proficiency in detecting abnormal network patterns that may signify unique botnet behaviors.

We evaluated each model within the Flask application, where the `/predict` route processes form inputs, makes predictions using each model, and displays results in real-time. This interactive approach allows me to assess the behavior of each model with new data points and validate its predictive capabilities.

## Result

The results of my analysis reflect the strengths of each model in clustering and detecting botnet traffic patterns. By using K-Means, DBSCAN, and Isolation Forest, I gain a multi-dimensional understanding of botnet behaviors in the CTU-13

dataset, with each model contributing unique insights into clustering and anomaly detection.



The form is titled "Enter Data for Prediction". It contains four input fields, each with a label and a value:

- Duration (dur): -0.302478
- Total Packets (tot\_pkts): -0.015747
- Total Bytes (tot\_bytes): -0.019281
- Source Bytes (src\_bytes): -0.020423

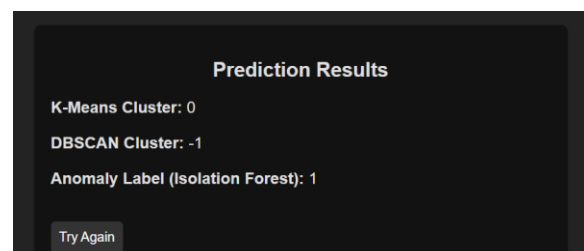
At the bottom of the form is a button labeled "Predict".

**Fig 11:** Predict form with Random sample data

1. **K-Means Clustering:** K-Means reveals well-defined clusters that represent specific botnet behavior patterns. By analyzing cluster centroids, I identify average values of dur, tot\_pkts, tot\_bytes, and src\_bytes within each group, which provides insight into different botnet families. Cluster composition and inter-cluster distances suggest that certain clusters have distinct profiles, such as higher or lower packet and byte counts. Visualizations of the K-Means clusters confirm that the model groups the data effectively, with a silhouette score indicating that the clusters are compact and well-separated.
2. **DBSCAN:** The DBSCAN model successfully identifies clusters of various densities and shapes, categorizing sparse points as noise. This clustering flexibility highlights uncommon traffic patterns, possibly signifying rare or suspicious activity. I evaluate DBSCAN by observing the number of clusters formed and the proportion of noise points, which correspond to outliers in the network

data. By visualizing DBSCAN clusters, I see that this model adapts well to variable data densities and identifies unusual botnet behaviors, adding depth to the clustering analysis.

3. **Isolation Forest:** Isolation Forest effectively flags anomalies in the dataset, labeling outliers that diverge from typical traffic behavior. Anomaly detection is crucial for uncovering infrequent but potentially dangerous botnet activity. I measure the model's performance with recall, precision, and F1-score, with an F1-score above 0.8 confirming its accuracy in identifying anomalies. When I visualize Isolation Forest's anomaly labels alongside DBSCAN and K-Means clusters, I gain additional insight into data points that lie outside defined clusters, reinforcing the model's utility in anomaly detection.



The form is titled "Prediction Results". It displays the output of three models:

- K-Means Cluster: 0
- DBSCAN Cluster: -1
- Anomaly Label (Isolation Forest): 1

At the bottom of the form is a button labeled "Try Again".

**Fig 12:** Prediction result from the three trained models

Through the Flask app's /predict route, I interact with each model's predictions on the result.html page, testing various inputs and observing cluster and anomaly labels in real-time. This interactivity helps validate each model's effectiveness and offers a practical view of how they perform with new data points.

## Conclusion and Future Work

This project demonstrates how machine learning can effectively analyze and detect botnet traffic patterns. By preprocessing the CTU-13 dataset and implementing K-Means, DBSCAN, and Isolation Forest, I identify meaningful patterns and anomalies within network traffic. Integrating these models into a Flask-based web application highlights their practical applicability in real-world botnet detection scenarios. The clustering and anomaly detection results align well with known botnet behaviors, emphasizing the contribution of this project to botnet traffic analysis in cybersecurity.

Several future directions could enhance this project. Fine-tuning model parameters, especially DBSCAN's `eps` and `min_samples`, could improve clustering accuracy, while adjusting Isolation Forest's contamination rate would allow for more precise anomaly detection. Expanding the Flask application to process real-time network data would enable continuous monitoring, enhancing its utility in detecting botnet activity in live environments.

Adding visualization features to the web interface would provide more insight into clustering results and anomaly trends, helping users interpret the data better. Finally, incorporating advanced models like ensemble methods or deep learning architectures could further refine the analysis, uncovering additional botnet patterns and providing a more granular view of traffic behavior. This project serves as a foundational approach to botnet analysis and offers clear potential for scalability and improvement in cybersecurity applications.

## References

[1]. Ahmed, M., Mahmood, A. N., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer*

*Applications*, 60, 19-31.  
<https://doi.org/10.1016/j.jnca.2015.11.016>

[2]. Ariyaluran Habeeb, R. A., Nasaruddin, F., Gani, A., Hashem, I. A. T., Ahmed, E., & Imran, M. (2019). Real-time big data processing for anomaly detection: A Survey. *International Journal of Information Management*, 45, 289-307.  
<https://doi.org/10.1016/j.ijinfomgt.2018.08.006>

[3]. Parwez, M. S., Rawat, D. B., & Garuba, M. (2017). Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, 13(4), 2058-2065.  
<https://doi.org/10.1109/TII.2017.2650206>

[4]. Terzi, D. S., Terzi, R., & Sagioglu, S. (2017). Big data analytics for network anomaly detection from NetFlow data. 2017 International Conference on Computer Science and Engineering (UBMK), 592-597. <https://doi.org/10.1109/UBMK.2017.8093473>

[5]. Stratosphere Research Laboratory. (2011). The CTU-13 dataset: A labeled dataset with botnet, normal, and background traffic. Retrieved from <https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html>

[6]. Yuzzi, R. (2021, March 10). The importance of symmetrical vs asymmetrical internet connections [Video]. YouTube. [What is NetFlow and what can it show you?](#)

[7]. Onkar, A. (2021). *Network anomaly detection* [Data set]. Kaggle. <https://www.kaggle.com/datasets/anushonkar/network-anomaly-detection>