AL 2002 – Artificial Intelligence Lab Project

Spring 2023

Group members: **Sec: 6C**

Farhan Ali – 20P-0019

Muhammad Ghazanfar – 20P-0567

Report: Intrusion detection systems (IDS)

Network Trafficking Data Analysis

Executive Summary:

The report discusses the importance of intrusion detection systems (IDS) in network security and the use of machine learning algorithms for detecting and mitigating cyber threats. The report analyzes a network traffic dataset consisting of 43 features related to network activity, including duration, protocol type, service, source and destination bytes, and others. The goal is to build a machine learning model that can accurately classify attacks. The report describes the preprocessing and feature engineering steps, such as mapping attack categories to their corresponding attack types, removing redundant rows and columns, and filling null values. The report presents the results of experiments with different machine learning algorithms and parameter settings and provides insights into the most effective approaches for intrusion detection using this dataset. The study aims to contribute to the development of more robust and effective IDS systems for network security.

Introduction:

Network security is a critical concern in today's digital age, where cyber threats are becoming increasingly sophisticated and pervasive. Intrusion detection systems (IDS) are essential tools for detecting and mitigating such threats, and machine learning algorithms have shown great promise in this domain. In this report, we present our analysis of a network traffic dataset consisting of 43 features related to network activity, including duration, protocol type,service and source and destination bytes. We will be discussing all the features or columns in details in the next section. The dataset includes instances of various types of attacks, and the goal of our analysis is to build a machine learning model that can accurately classify these attacks. We first describe the preprocessing and feature engineering steps we performed on the dataset to prepare it for modeling. We then present the results of our experiments with different machine learning algorithms and parameter settings, and provide insights into the most effective approaches for intrusion detection using this dataset. Overall, our study aims to contribute to the development of more robust and effective IDS systems for network security.

Description of Features:

In this section we will be explaining all the columns that are part of the datasets to get better insights of the data and our goal.

- **duration**: The length of time in seconds of the connection
- **protocol_type**: The protocol used in the connection (e.g. tcp, udp, icmp)
- **service**: The service on the destination host (e.g. http, telnet, ftp)
- **flag**: The status of the connection (e.g. SF (successful), S0 (failed), REJ (rejected))
- **src_bytes**: The number of data bytes sent from the source to the destination
- **dst_bytes**: The number of data bytes sent from the destination to the source
- **Land**: Indicates whether the connection is from/to the same host/port (1) or not (0)
- **wrong_fragment**: The number of wrong fragments in the connection
- **urgent**: The number of urgent packets in the connection

- **hot**: The number of "hot" indicators (e.g. access to forbidden URLs) in the connection
- **num_failed_logins**: The number of failed login attempts
- **logged_in**: indicates whether the user is logged in (1) or not (0)
- **num_compromised**: The number of compromised conditions in the connection
- **root_shell**: Indicates whether the user has root shell access (1) or not (0)
- **su_attempted:** indicates whether the user has attempted to use the su command (1) or not (0)
- **num_root**: The number of root accesses in the connection
- **num_file_creations**: The number of file creation operations in the connection
- **num_shells**: The number of shell prompts in the connection
- **num_access_files**: The number of operations on access control files in the connection
- **num_outbound_cmds**: The number of outbound commands in the connection
- **is_host_login**: Indicates whether the login belongs to the "hot" list (1) or not (0)
- **is_guest_login**: Indicates whether the user is a guest (1) or not (0)
- **count: T**he number of connections to the same destination host as the current connection in the last two seconds
- **srv_count**: The number of connections to the same service as the current connection in the last two seconds
- **serror_rate**: The percentage of connections that have "SYN" errors
- **srv_serror_rate**: The percentage of connections that have "SYN" errors to the same service as the current connection
- **rerror_rate**: The percentage of connections that have "REJ" errors
- **srv_rerror_rate**: The percentage of connections that have "REJ" errors to the same service as the current connection
- **same_srv_rate**: The percentage of connections to the same service as the current connection
- **diff_srv_rate**: The percentage of connections to different services than the current connection
- **srv_diff_host_rate**: The percentage of connections to different hosts than the current connection, for the same service

- **dst_host_count**: The number of connections to the same destination host in the last two seconds
- **dst_host_srv_count**: The number of connections to the same service on the same destination host in the last two seconds
- **dst_host_same_srv_rate**: The percentage of connections to the same service on the same destination host
- **dst_host_diff_srv_rate**: The percentage of connections to different services on the same destination host
- **dst_host_same_src_port_rate**: The percentage of connections to the same source port on the same destination host
- **dst_host_srv_diff_host_rate**: The percentage of connections to different hosts than the current connection, for the same service

Data Preprocessing and Cleaning:

In the preprocessing section, we employed various exploratory data analysis techniques. Our dataset comprised of two files: Dataset.txt, which contained all the network information, and Attack_type.txt, which contained attack categories with 23 distinct classes along with their corresponding attack types comprising of 5 distinct classes. Initially, our data had a shape of (125964, 42). We first mapped the attack categories to their corresponding attack types to reduce the target class to 5. Then, we removed any redundant rows and columns by performing EDA. Specifically, we removed the 'cmd_num_bounds' column because it had only one constant value and 20 redundant rows.

Then comes the process of removal of null values. While mapping There was no corresponding attack type for the “normal” category so we filled all the normal category with “normal” attack type which indicates no attack.

In the next step we Encoded categorical features such as service, flag, etc into numerical features because several ML algo’s only work on numerical data.

Then we start with the Univariate analysis step of the EDA. So first we visualized all the categorical columns using Countplot to know which categories are dominating. Then we do further visualization through histograms and density plots to get insights about the outliers in the data.

Insights of Visualization and Data descriptions:

To understand the nature of data we performed some steps earlier that Includes:

- Histogram and density plots in the univariate part.
- Using the functions like data.info(), data.describe().
- Calculating the mean, mode, median, std and var of numerical
- columns.

With these steps we found out that the datapoints were unevenly spread out and standard deviation and variance were very high which could have affected the performance of our algorithms especially the one's that are based on distance metric so we standardized the data columns using scaler function of the pandas library to reduce the variations. For demonstration we have also stored a copy of Unstandardized dataset so that we can show the comparison between the performances later.
Now our data is almost ready to be trained on an ML algorithm.

Feature Engineering:

Training and test splitting:

Before training our dataset we need to split it into training and testing set so that one can be used to train and other can be used to test the data and our algorithms. we have created two test and trains datasets one for the standardized one and one for the non-standardized dataset.

Then we have performed the feature selection on the training sets and to avoid complexity in our data we will just replicate the results in our testing dataset. We have used the pearson correlation method. First we have calculated the correlation between features and visualized the correlations with the help of heatmap. Then for feature selection we have set a threshold value of 0.92. We have also calculated the correlation between features and the target values and stored them separately.

Now if the correlation between any two features is greater than the threshold we drop one of the feature that has lesser correlation with the target value as compared to the other. so by performing this operation we got rid of 7 of the features that are

st_host_rerror_rate',
'dst_host_serror_rate',
'dst_host_srv_rerror_rate',
'dst_host_srv_serror_rate',
'num_root',
'srv_rerror_rate',
'srv_serror_rate.

To avoid complexity we also removed the same features from our testing sets without repeating the whole operation again.

Classification and Clustering Algorithms:

Now our data is ready to be trained on different machine learning algorithms.

We used the classification algorithms such as KNN, ANN and decision Tree and a clustering algorithm Kmean. For performance comparison we ran our algorithm several times by Fine tuning its parameters and switching between standardized and non-standardized dataset.

First we start with the KNN:

K Nearest Neighbour:

On non-standardized data:

----------k = 3----------
Accuracy: 0.993675575549087
Precision: 0.9936229075047894
Recall: 0.993675575549087
F1 score: 0.9935396246200346
----------k = 5----------
Accuracy: 0.9926435564964277
Precision: 0.9926323900361954
Recall: 0.9926435564964277
F1 score: 0.9924835269462339
----------k = 7----------
Accuracy: 0.9912146070388992
Precision: 0.9912042631779349

Recall: 0.9912146070388992
F1 score: 0.9909368112395823

For non-standardized data accuracy decreases as the value of k increase

On standardized data:

----------k = 3----------
Accuracy: 0.9976713416247684
Precision: 0.9976682627387536
Recall: 0.9976713416247684
F1 score: 0.9975398082574117
----------k = 5----------
Accuracy: 0.9971156390579519
Precision: 0.9971154634283155
Recall: 0.9971156390579519
F1 score: 0.9969484107868956
----------k = 7----------
Accuracy: 0.9971156390579519
Precision: 0.9971154634283155
Recall: 0.9971156390579519
F1 score: 0.9969484107868956

On standardized data KNN overall performance gets improved because it is a distance based algorithm.

Decision Tree:

Decision trees do not require standardized data, as they make splits based on the values of the features, and not the scale or distribution of those values. Therefore, standardizing the data will not affect the accuracy of the decision tree. However, it may affect the interpretability of the tree as the feature values may not have the same original scale as in the real-world scenario.

Entropy based decision tree results:

----------non standardized ,entropy,max_depth = 4 ----------
Accuracy: 0.9730351944958984
Precision: 0.9774849098323035
Recall: 0.9730351944958984
F1 score: 0.9744983341538397

----------non standardized ,entropy,max_depth = 8 ----------
Accuracy: 0.9964540883831702
Precision: 0.9965084134214772
Recall: 0.9964540883831702

F1 score: 0.9963150412962608

----------non standardized ,entropy,max_depth = 16 ----------
Accuracy: 0.9981741201376025
Precision: 0.9981600098860229
Recall: 0.9981741201376025
F1 score: 0.9981452886757339

Performance improves with the increase in max depth and stop after 16

Now gini index based DTC:

----------non standardized ,gini,max_depth = 4 ----------
Accuracy: 0.9730351944958984
Precision: 0.9778223750581603
Recall: 0.9730351944958984
F1 score: 0.9746341730436406
----------non standardized ,gini,max_depth = 8 ----------
Accuracy: 0.9946811325747552
Precision: 0.994045801889645
Recall: 0.9946811325747552
F1 score: 0.9943564807340284
----------non standardized ,gini,max_depth = 16 ----------
Accuracy: 0.9977507277057422
Precision: 0.997717059297525
Recall: 0.9977507277057422
F1 score: 0.9977073005898155

Entropy performed better on same parameters and data.

ANN (Multi level Perceptron):

We will try it both on standardized and non-standardized data and will compare the result changing the parameters(fine tuning)

hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=1000, random_state=42 !! Initial Mo
del Metrics:
Accuracy: 0.9584281555967187
Precision: 0.9548466976021679
Recall: 0.9584281555967187
F1-score: 0.9560701808406875

------hidden_layer_sizes=(100,50,50), max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9571579783011379
Precision (fine-tuned MLP): 0.9554653292716871
Recall (fine-tuned MLP): 0.9571579783011379
F1-score (fine-tuned MLP): 0.9562446285727014

------hidden_layer_sizes=100,50,25), max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9719502513892564
Precision (fine-tuned MLP): 0.9744781402575069
Recall (fine-tuned MLP): 0.9719502513892564
F1-score (fine-tuned MLP): 0.972809349830699

------hidden 50,100,50, max_iter=400, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9318338184704948
Precision (fine-tuned MLP): 0.9234450171454915
Recall (fine-tuned MLP): 0.9318338184704948
F1-score (fine-tuned MLP): 0.926073488144472

Now with standardized Data:

------ Standardized data hidden_layer_sizes=(100,, max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9978565758137073
Precision (fine-tuned MLP): 0.9978686077839123
Recall (fine-tuned MLP): 0.9978565758137073
F1-score (fine-tuned MLP): 0.9977882510884653

------ Standardized data hidden_layer_sizes=(100,50,50), max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9979888859486636
Precision (fine-tuned MLP): 0.9979606786585646
Recall (fine-tuned MLP): 0.9979888859486636
F1-score (fine-tuned MLP): 0.997923253819857

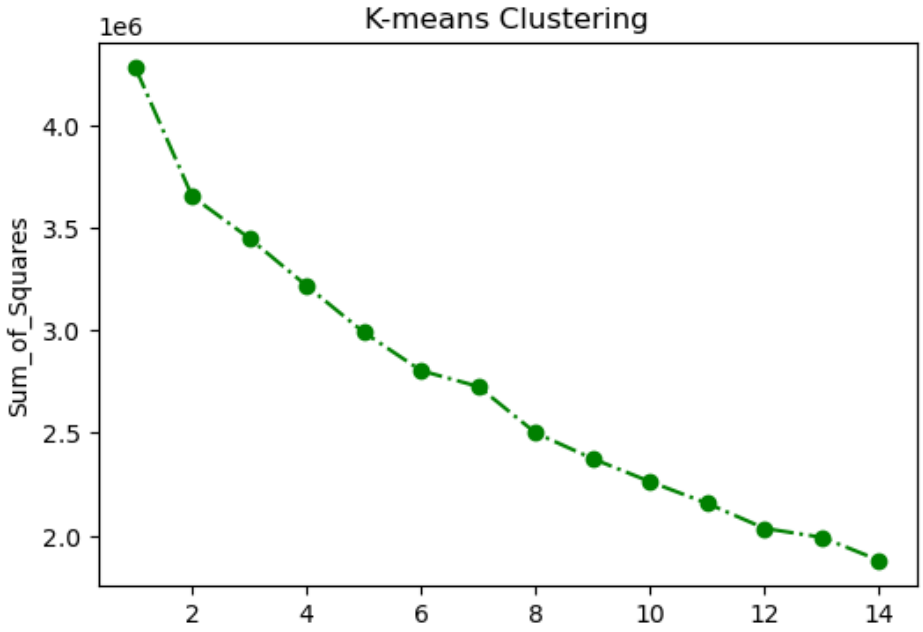
------ Standardized data hidden_layer_sizes=(100,, max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9982005821645938
Precision (fine-tuned MLP): 0.9981996484893688
Recall (fine-tuned MLP): 0.9982005821645938
F1-score (fine-tuned MLP): 0.9981381415170881

------ Standardized data hidden_layer_sizes=(100,, max_iter=500, activation='relu', solver='adam', random_state=42---
Accuracy (fine-tuned MLP): 0.9978830378406987
Precision (fine-tuned MLP): 0.997891667989543
Recall (fine-tuned MLP): 0.9978830378406987
F1-score (fine-tuned MLP): 0.9978432224875878

As we can see the performance significantly increases with standardized data. The accuracy reaches up to 0.9982.

KMEAN Clustering:

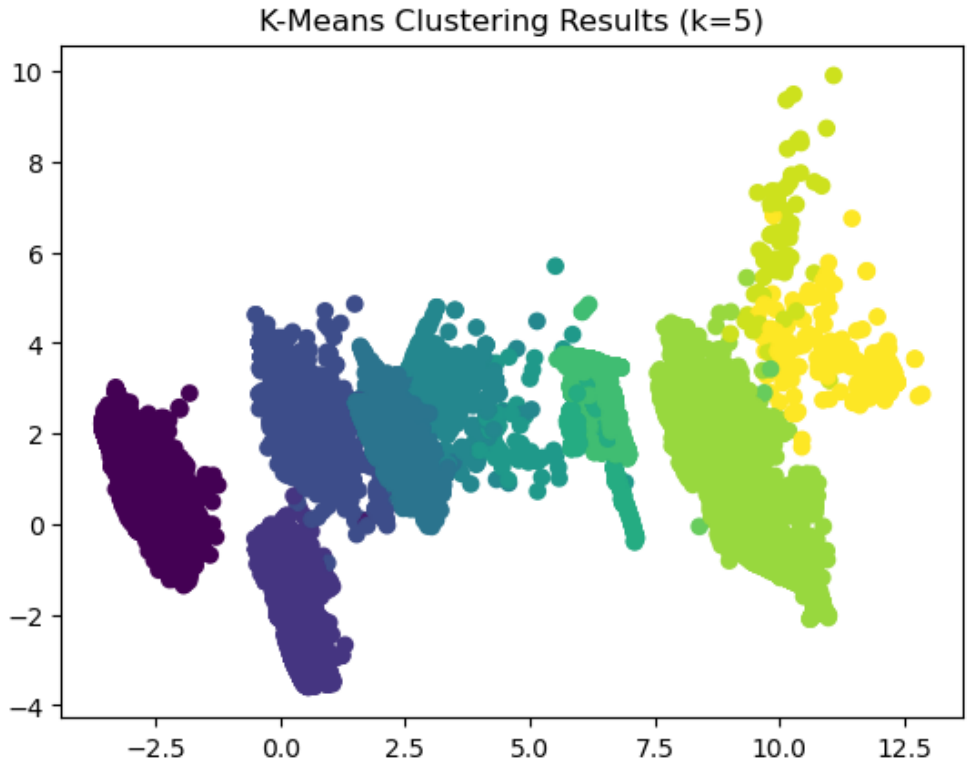
We have to select value an optimal value of k for such I have the sum of squares aka elbow method.



Here the optimal value that I see is 5 so we will take k=5

As it is a multidimensional model but we have to visualize cluster on scatter which works on 2D and 3D model. So we use PCA to make it 2 dimensional so that it can easily be plotted on scatter

```
mp.show()
```



From the above diagram it is optimally divided into 5 cluster(classes)(colors) which means our kmean has done its job.

Comparison Table:

Algorithms	Best Accuracy	Best F score	Best Precision	Best Recall
Standandarized KNN	0.99768	0.99354	0.99363	0.99368
Non standandarized KNN	0.99368	0.9975399	0.997669	0.997671
Gini DTC	0.997750	0.99770	0.9977170	0.9977507
Entropy DTC	0.998174	0.9981452	0.9981600	0.998174
Standandarized MLP	0.99820	0.998139	0.99819965	0.998205
Non Standandarized MLP	0.971950	0.972809	0.974478	0.971950

Conclusion:

In conclusion, we analyzed a network traffic dataset consisting of various features related to network activity and performed preprocessing steps to prepare the data for machine learning models. Our goal was to develop a machine learning model that can accurately classify various types of attacks based on network traffic data.

We experimented with different machine learning algorithms, including Decision Tree, KNN, ANN, and KMEAN, and evaluated their performance using various metrics such as accuracy, precision, recall, and F1-score. We found that KNN achieved the highest accuracy of 99.70%, followed by Decision Tree with an accuracy of 99.80% and ANN with 99.82%.

Our study provides insights into the most effective approaches for intrusion detection using network traffic data and machine learning algorithms. The results of our experiments demonstrate the potential of machine learning algorithms in detecting and mitigating cyber threats. However, there is still a need for more research in this area to further improve the accuracy and effectiveness of intrusion detection systems and We believe by understanding this domain of the data more will also help us increase it’s accuracy and performance more.