Detecting Network Intrusions

Data Analysis and Preparation

January Johnson

March 3, 2022

Contents

1 Introduction

An intrusion detection system (IDS) is an important cyber security technique, which monitors the state of software and hardware running in the network. Existing IDSs still confront hurdles in increasing detection accuracy, lowering false alarm rates, and identifying novel assaults, despite years of scientific research.Cyber security techniques include anti-virus software, firewalls and intrusion detection systems (IDSs), as well as threat-based approaches such as phishing and identity theft. Another issue with current IDSs is that they are incapable of detecting unknown assaults. Because network environment evolve continuously, new attack types and threats occur on a regular basis. As a result, IDSs that can identify unknown threats must be developed.

Researchers have begun to focus on constructing IDSs that use machine learning approaches to extract meaningful data from enormous databases automatically. Machine learning-based IDS may attain greater detection levels when adequate training data is available, and models have enough generalizability to recognize attack variants and unique threats. They are also easier to develop and build because they do not depend significantly on specialized knowledge.

The Problem

Current implementations of AI within cybersecurity for intrusion detection are either very good at detecting intrusions that are already known (that is finding well known and documented signatures or attack patterns within a network alerting the system and preventing the attack from moving forward) or have the ability to detect unknown attacks requiring very little information on previous attack patterns and not requiring a large database of attack signatures to do so. In both cases, one implementation gives rise to threats by unknown attacks and requires a large database to maintain and the other has a high false alarm rate. Considering the field of cybersecurity is at least forty years old and the use of AI (in one manifestation or another) is nearing two decades the current state of IDS leaves much to be desired not to mention a smoldering questions of why and how. That is why haven't detection systems grown beyond their current underwhelming limitations and how can AI be used (really just

ML) be used to improve IDS systems. With these questions and challenges in mind AI professionals are turning my to either deep learning or new techniques of combining multiple types of machine learning algorithms (primarily classifications) to improve IDS systems.

Proposed Solution - Project Plan.

Flow-Based Attack Detection and Feature Engineering-Based Detection I intend to use a features vectors algorithms to do flow-based detection. In comparison to other approaches with flow detection prepossessing is simple since it requires no packet parsing or session reorganization.

Notebook

2 Dataset

The "KDD Cup 1999: Computer network intrusion detection" dataset was obtained from the KDD Cup 1999: Computer network intrusion detection – SIGKDD website. KDD99 is the most used IDS benchmark dataset. Its compilers mined DARPA 1998 data for 41-dimensional characteristics. Basic features, content features, host-based statistical features, and time-based statistical features are the four categories of features in KDD99. The data set consists of various features to identify whether traffic is a type of attack or normal. The target class consists of different types of attacks smurf, neptune, back, satan, ipsweep, portsweep, and teardrop. The dataset contains 41 features (independent variables) that are used to categorize the traffic into different types as indicated by the status variable (dependent variable). There are 494,021 rows of data consisting of over 300,000 duplicates. The following is the list of features:

duration: continuous.
protocol_type: symbolic.
service: symbolic.
flag: symbolic.
src_bytes: continuous.
dst_bytes: continuous.

land: symbolic.
wrong_fragment: continuous.
urgent: continuous.
hot: continuous.
num_failed_logins: continuous.
logged_in: symbolic.
num_compromised: continuous.
root_shell: continuous.
su_attempted: continuous.
num_root: continuous.
num_file_creations: continuous.
num_shells: continuous.
num_access_files: continuous.
num_outbound_cmds: continuous.
is_host_login: symbolic.
is_guest_login: symbolic.
count: continuous.
srv_count: continuous.
serror_rate: continuous.
srv_serror_rate: continuous.
rerror_rate: continuous.
srv_rerror_rate: continuous.
same_srv_rate: continuous.
diff_srv_rate: continuous.
srv_diff_host_rate: continuous.
dst_host_count: continuous.
dst_host_srv_count: continuous.
dst_host_same_srv_rate: continuous.
dst_host_diff_srv_rate: continuous.
dst_host_same_src_port_rate: continuous.
dst_host_srv_diff_host_rate: continuous.
dst_host_serror_rate: continuous.
dst_host_srv_serror_rate: continuous.
dst_host_rerror_rate: continuous.
dst_host_srv_rerror_rate: continuous.

2.1 Visualization of the distribution of key input features

Because the data set consist of 41 features a correlation matrix was used to identify features with strong relation to the outcome. As you from the correlation Matrix, we can see that the flag feature is highly correlated to the srv_serror_rate, the logged_in feature is highly correlated to the dst_host_same_srv_rate, the hot feature is highly correlated to the is_guest_login feature, and so on.

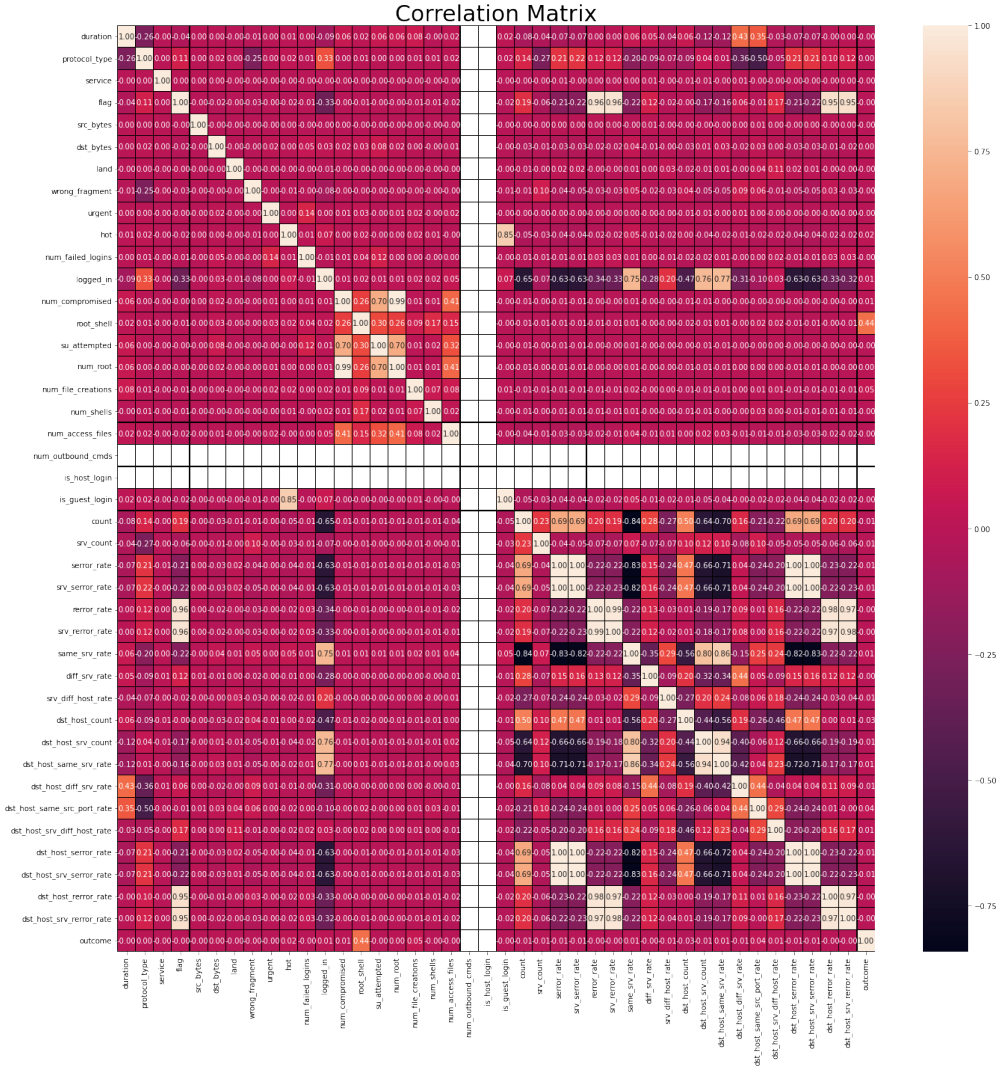

Figure 1: Correlation Matrix

The graphics below illustrate the histogram plot of each correlated feature's data focus, which highlights their most high and minimum values, as well as how they are dispersed.

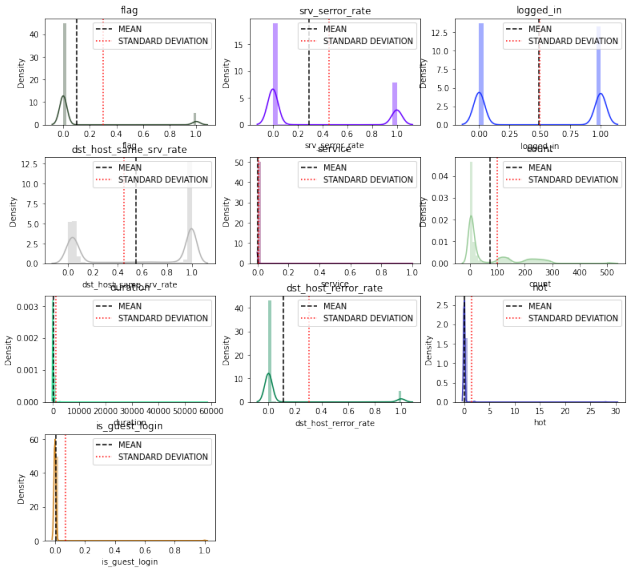

Figure 2: Correlated Features Data Distribution Histograms

	flag	srv_serror_rate	logged_in	dst_host_same_srv_rate	service	count	duration	dst_host_rerror_rate	hot	is_guest_login
count	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000	145586.000000
mean	0.101054	0.291551	0.491490	0.553222	0.000076	74.385593	132.025181	0.110463	0.100174	0.004705
std	0.301401	0.453567	0.499929	0.456236	0.008692	100.335945	1224.157053	0.306097	1.426798	0.068433
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.050000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.770000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	1.000000	0.000000	132.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	511.000000	58329.000000	1.000000	30.000000	1.000000

Figure 3: Correlated Feature Statistics

2.2 Distribution of the outcome feature

The target is unbalanced due to an oversampling of typical network traffic. Data augmentation (e.g., resampling, undersampling, or producing generated data) will be necessary in the future.

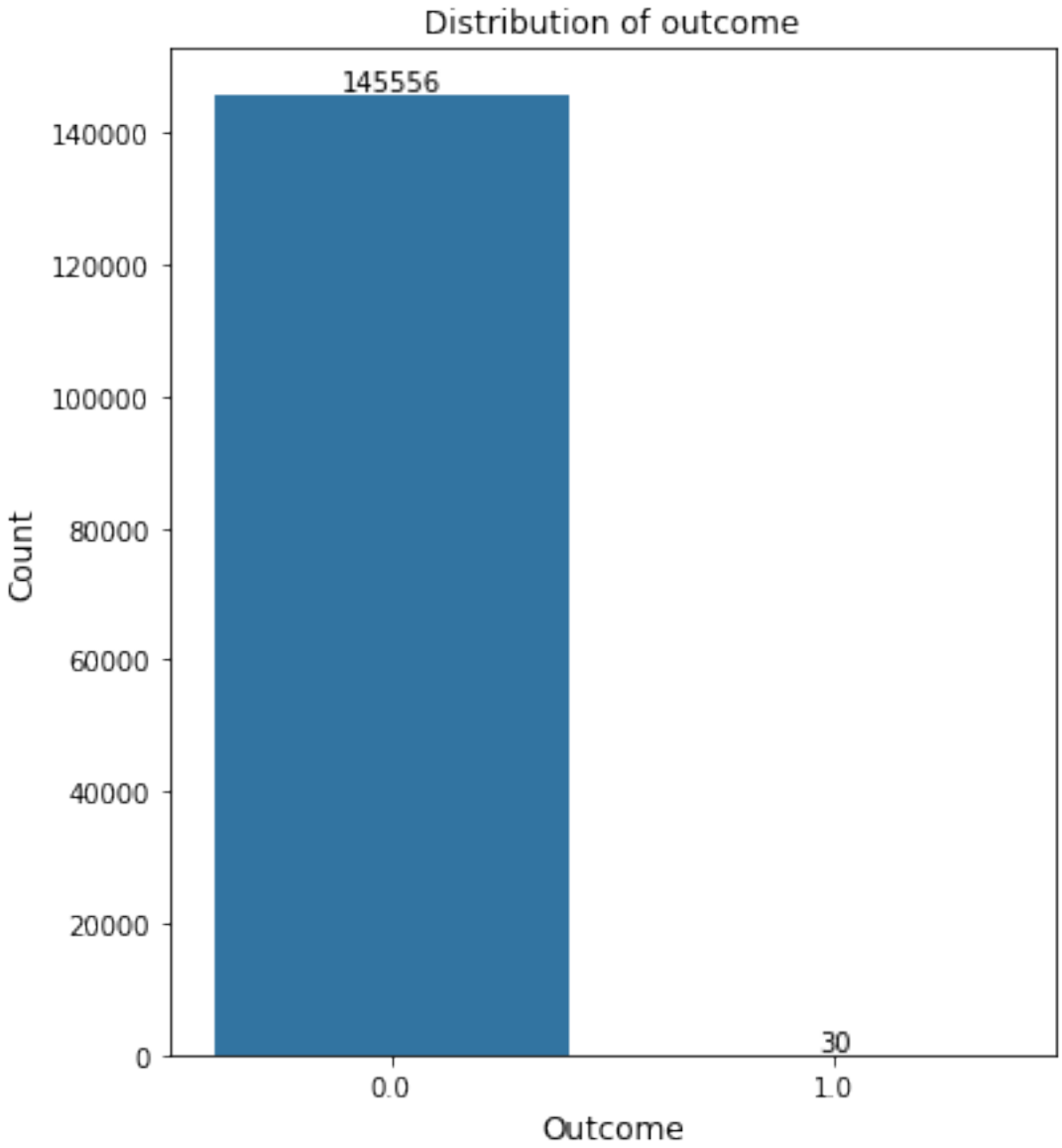

Figure 4: Distribution of Outcome

3 Data Processing

3.1 Data Normalization

It's vital to normalize the characteristics of a dataset to accurately analyze data that often uses varying units of measurement. Parameters recorded at various scales do not contribute uniformly to the analysis and can potentially create a basis. To normalize a data structure, all of the components must be between 0 and 1. This aligns all of the numeric column values in the dataset to the same scale. The most common way to normalize a vector is to divide it by its norm. It might also involve re-scaling by the lowest and maximum values. I chose to re-scale using highest and lowest values.

Normalization by Lowest and Maximum

$$zi = \frac{Xi - X_{min}}{X_{max} - X_{min}} \tag{1}$$

Normalization by the Mean

$$zi = \frac{X - X_{mean}}{X_{max} - X_{min}} \tag{2}$$

3.2 Normalized Data

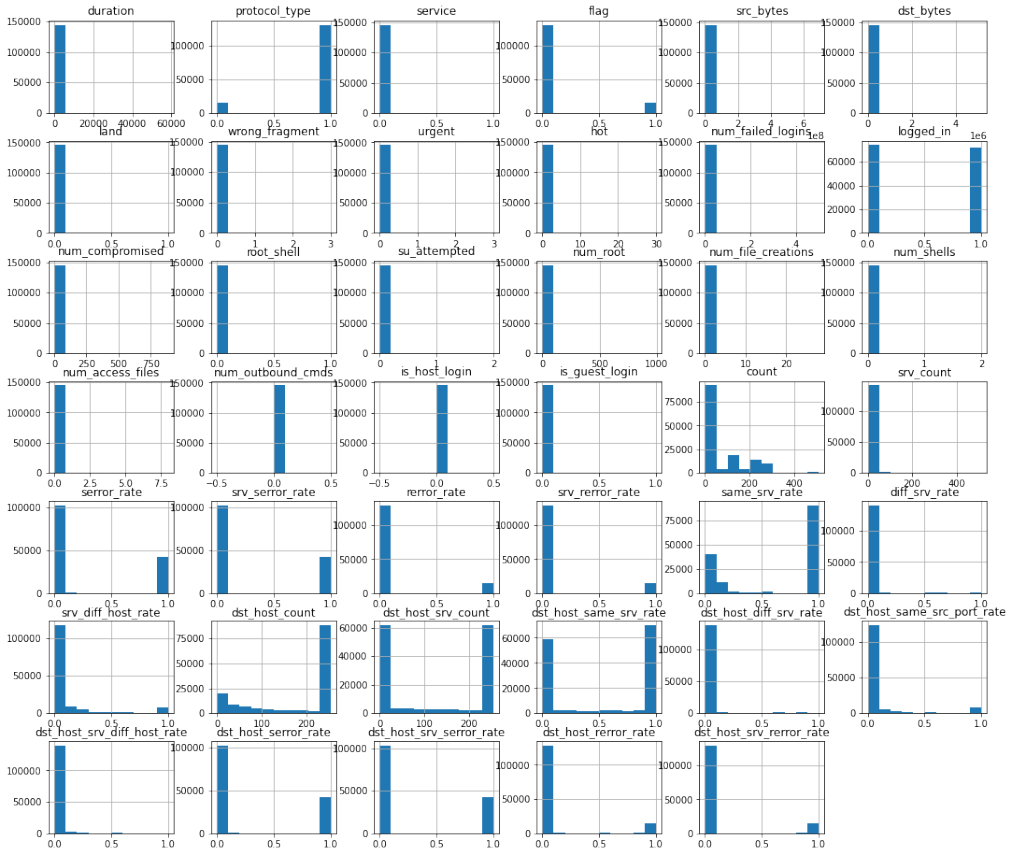

Figure 5: Normalized Data (Min-Max)