October 5, 2021

Predicting Anomalies in Network Traffic

Phase-1 : Data Analysis and Preparation
Phase-2 : Build a model to overfit the dataset

Reshma Priya, Manne Muddu
AI 5300
University of Missouri, St Louis, Fall 2021

Contents

1 Introduction

1.1 Intrusion Detection Systems (IDS)

Intrusion Detection Systems (IDS) provide network security software applications by continuously monitoring network traffic and classifying the connections as normal or malicious. IDS are categorized into two types based on the responsive nature - Passive IDS and Active IDS. A passive IDS is designed to identify and block the malicious and malware attacks manually by human experts whereas active IDS is designed to identify and block the malware attacks using a software automatically.

IDS are also categorized into Signature based IDS and Anomaly Based IDS. In the Signature based IDS, there exists a database which contains details about all known malware attacks against which each network traffic connection is validated to identify the malicious nature if it exists. This type of IDS is costly and has to keep updating new types of attacks frequently. Anomaly based IDS is a behavior-based system where any deviation from the normal network traffic patterns will be reported using pattern-recognition techniques. In this research paper, a proof-of-concept for a machine learning based Anomaly based intrusion detection will be evaluated on a benchmark intrusion detection dataset.

1.2 Motivation

The Internet of Things (IoT) has changed the way devices communicate across a network. There are over 30 billion devices connected to each other in today's world where their communication happens over the network. Due to the variety, velocity and vast nature of this traffic, the traditional intrusion detection systems are not sufficient to identify the malware patterns [1]. With the help of data science, machine learning and neural network techniques, one can leverage the historic network traffic patterns and can build an intelligent artificial intelligence model to solve this intrusion detection problem. This project exactly tries to provide a proof-of-concept for this problem by applying machine learning and neural network techniques on a benchmark network intrusion detection dataset.

1.3 Problem Statement

In this research, KDD Cup 1999 dataset will be used to build a machine learning based intrusion detection problem to predict (classify) whether a network connection is "normal" or "abnormal". As this is a categorical prediction, this is a binary classification problem

Github Link to this project:click here
Jupyter notebook html: click here

2 Dataset Description and Exploration

In this research, the KDD Cup 1999 dataset has been chosen for the data analysis and model building. This dataset was used in the fifth international conference for the data mining and knowledge discovery competition and contains a huge variety of network intrusion data that is simulated in an environment equipped with a military network setting.

The original source of the dataset is from the official KDD website [2] and can also be found in the kaggle website [3]. The dataset contains about 494,000 records and 41 features (columns) which contains network traffic details like source bytes, destination connection information, type of attacks and so on. Below are the list of features (columns) and their data types in the dataset.

2.1 Target Variable "label" distribution

The target variable "label" contains all different types of malware attacks and also the value "normal" (i.e., not an attack) as shown below.

'normal', 'buffer overflow', 'loadmodule', 'perl', 'neptune', 'smurf', 'guess passwd', 'pod', 'teardrop', 'portsweep', 'ipsweep', 'land', 'ftp write', 'back', 'imap', 'satan', 'phf', 'nmap', 'multihop', 'warezmaster', 'warezclient', 'spy', 'rootkit'

All malware attacks are grouped (transformed) to the value "abnormal" to make this problem a binary classification problem instead of a multi-class classification. The target variable distribution can be found below (see **Figure 1**).

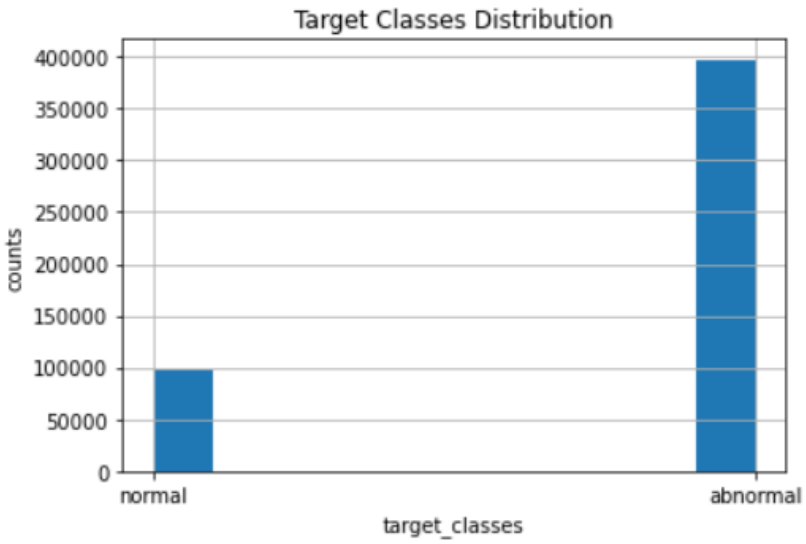


Figure 1: Target Variable "label" Distribution.

2.2 Categorical Columns Distributions

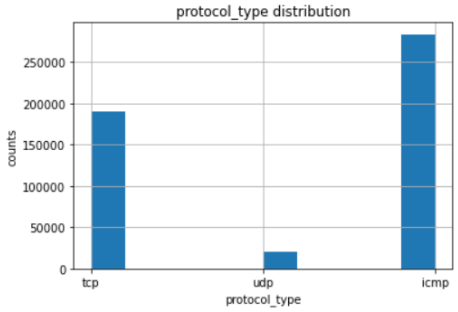
Below are the categorical columns in this dataset and their corresponding distribution plots.

'protocol type', 'service', 'flag', 'land', 'logged in', 'is host login', 'is guest login'

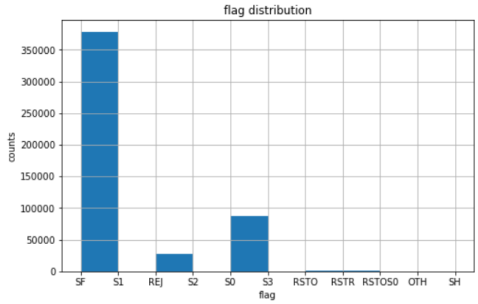
Categorical variables statistics can be found in the below table:

Table 1: Categorical Variables Statistics.

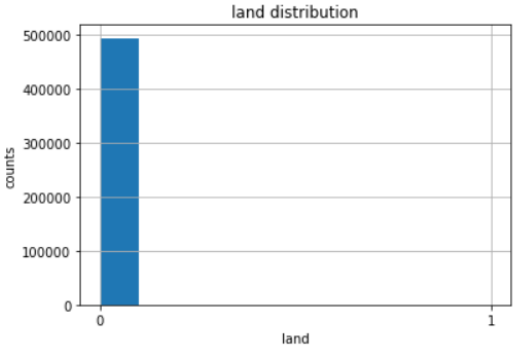
Stats	protocol_type	service	flag	land	logged_in	is_host_login	is_guest_login
count	494020	494020	494020	494020	494020	494020	494020
unique	3	66	11	2	2	1	2
top	icmp	ecr_i	SF	0	0	0	0
freq	283602	281400	378439	493998	420784	494020	493335



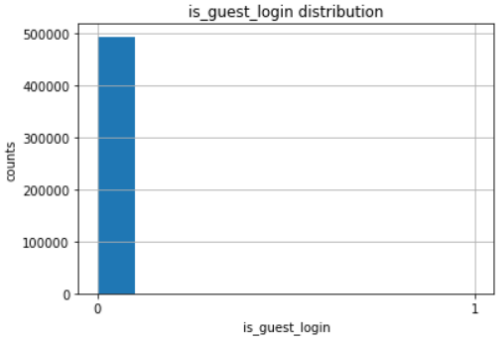
(a) protocol type



(b) Flag

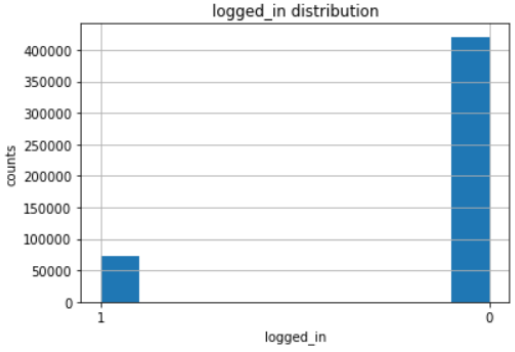


(c) Land



(d) Is Guest Login

(e) Is Host Login



(f) Logged In

Figure 2: Categorical Columns Distributions

2.3 Continuous Columns

Below are the continuous columns in this dataset and their corresponding distribution plots.

'duration', 'src bytes', 'dst bytes', 'wrong fragment', 'urgent', 'hot', 'num failed logins', 'num compromised', 'root shell', 'su attempted', 'num root', 'num file creations', 'num shells', 'num access files', 'num outbound cmds', 'count', 'srv count', 'serror rate', 'srv serror rate', 'rerror rate', 'srv rerror rate', 'same srv rate', 'diff srv rate', 'srv diff host rate', 'dst host count', 'dst host srv count', 'dst host same srv rate', 'dst host diff srv rate', 'dst host same src port rate', 'dst host srv diff host rate', 'dst host serror rate', 'dst host srv serror rate', 'dst host rerror rate', 'dst host srv rerror rate'

Continuous variables statistics can be found in the below table:

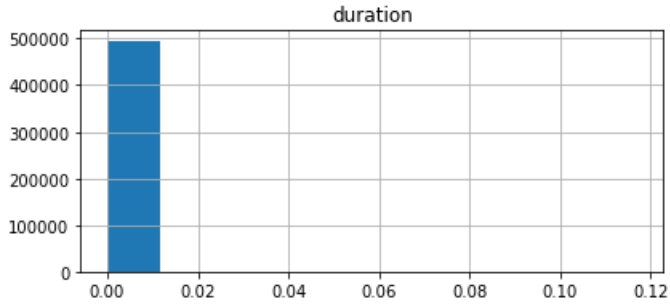
Table 2: Categorical Variables Statistics.

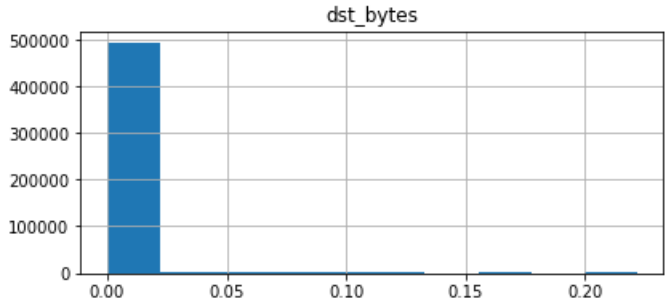
Stats	count	mean	std	min	25%	50%	75%	max
duration	494020	47.9794	707.7472	0	0	0	0	58329
src_bytes	494020	3025.616	988219.1	0	45	520	1032	6.93E+08
dst_bytes	494020	868.5308	33040.03	0	0	0	0	5155468
wrong_fragment	494020	0.006433	0.134805	0	0	0	0	3
urgent	494020	1.42E-05	0.00551	0	0	0	0	3
hot	494020	0.034519	0.782103	0	0	0	0	30
num_failed_logins	494020	0.000152	0.01552	0	0	0	0	5
num_compromised	494020	0.010212	1.798328	0	0	0	0	884
root_shell	494020	0.000111	0.010551	0	0	0	0	1
su_attempted	494020	3.64E-05	0.007793	0	0	0	0	2
num_root	494020	0.011352	2.01272	0	0	0	0	993
num_file_creations	494020	0.001083	0.096416	0	0	0	0	28
num_shells	494020	0.000109	0.01102	0	0	0	0	2
num_access_files	494020	0.001008	0.036482	0	0	0	0	8
num_outbound_cmds	494020	0	0	0	0	0	0	0
count	494020	332.2864	213.1471	0	117	510	511	511
srv_count	494020	292.9071	246.3227	0	10	510	511	511
serror_rate	494020	0.176687	0.380717	0	0	0	0	1
srv_serror_rate	494020	0.176609	0.381017	0	0	0	0	1
rerror_rate	494020	0.057434	0.231624	0	0	0	0	1
srv_rerror_rate	494020	0.057719	0.232147	0	0	0	0	1
same_srv_rate	494020	0.791547	0.38819	0	1	1	1	1
diff_srv_rate	494020	0.020982	0.082206	0	0	0	0	1
srv_diff_host_rate	494020	0.028996	0.142397	0	0	0	0	1
dst_host_count	494020	232.4712	64.7446	0	255	255	255	255
dst_host_srv_count	494020	188.6661	106.0402	0	46	255	255	255
dst_host_same_srv_rate	494020	0.753781	0.41078	0	0.41	1	1	1
dst_host_diff_srv_rate	494020	0.030906	0.109259	0	0	0	0.04	1
dst_host_same_src_port_rate	494020	0.601936	0.481309	0	0	1	1	1
dst_host_srv_diff_host_rate	494020	0.006684	0.042133	0	0	0	0	1
dst_host_serror_rate	494020	0.176754	0.380593	0	0	0	0	1
dst_host_srv_serror_rate	494020	0.176443	0.38092	0	0	0	0	1
dst_host_rerror_rate	494020	0.058118	0.23059	0	0	0	0	1
dst_host_srv_rerror_rate	494020	0.057412	0.230141	0	0	0	0	1

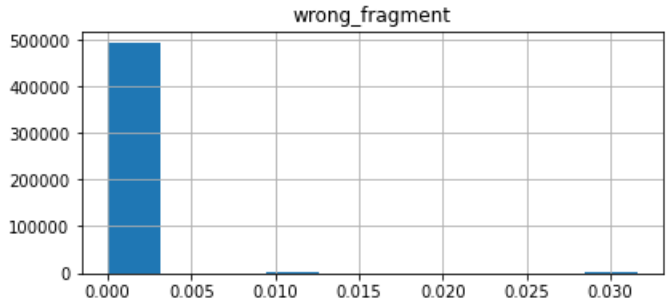
3 Data Normalization

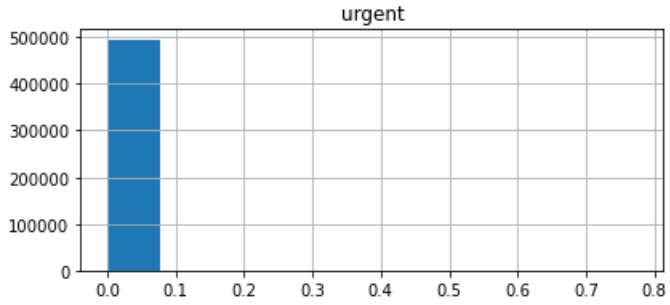
As seen from the above continuous variables distribution in the above section the scale of some of the features (columns) values are not in same range as others. This irregularity will make the machine learning model train poorly. Hence the features need to be normalized so that the optimization during model training will happen in a better way and the model will not be sensitive to the features. This research paper uses the existing normalization function that exists in python's scikit-learn library which can be found here

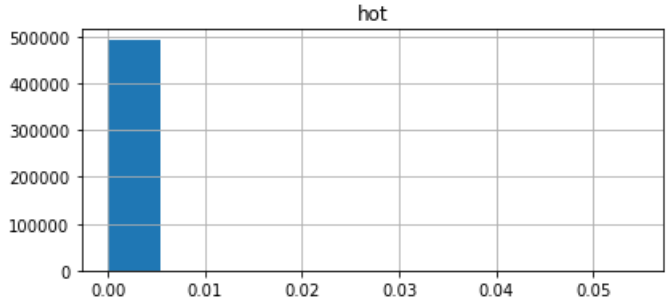
After applying normalization transformation to each continuous feature, below are the new distribution plots.

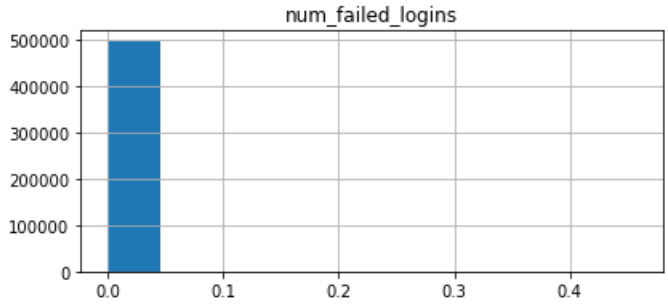


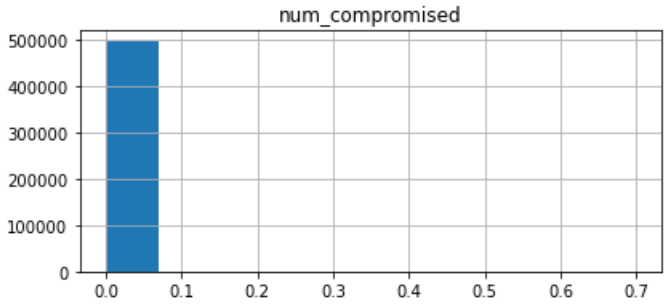


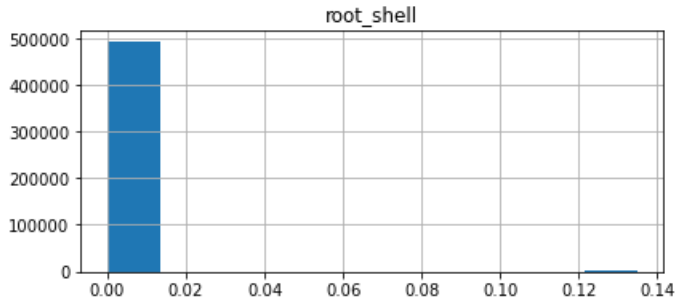


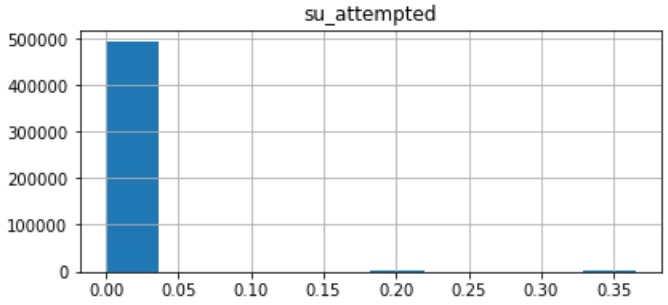


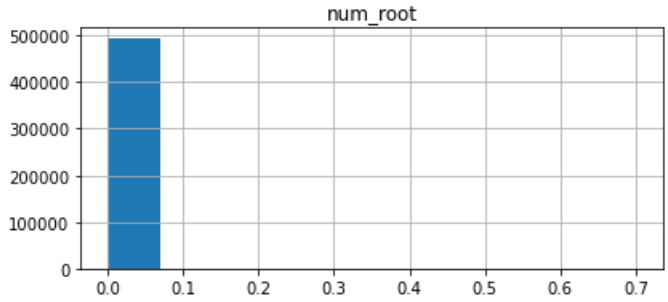


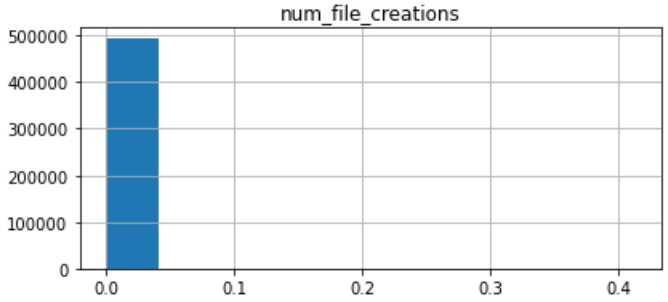

num_failed_logins
500000
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4

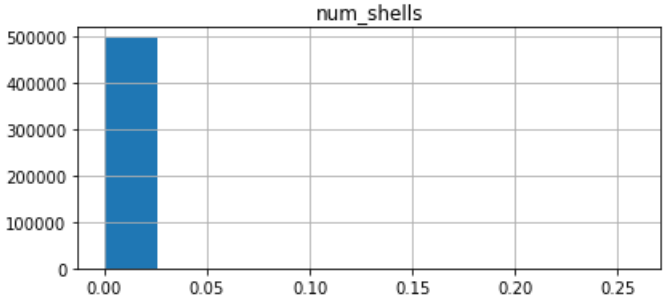

num_compromised
500000
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

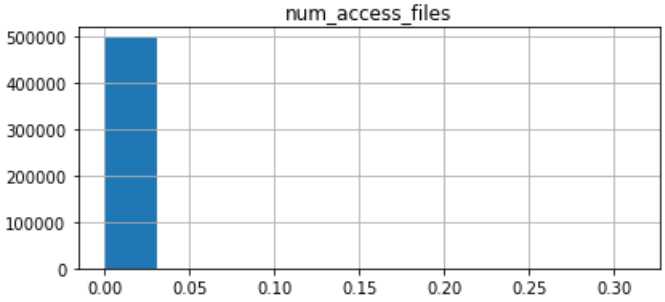

root_shell
500000
400000
300000
200000
100000
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

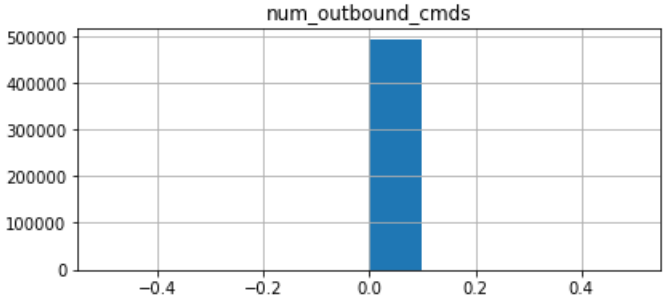

su_attempted
500000
400000
300000
200000
100000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35

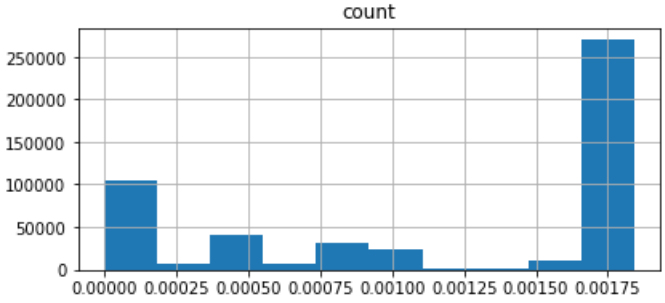

num_root
500000
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

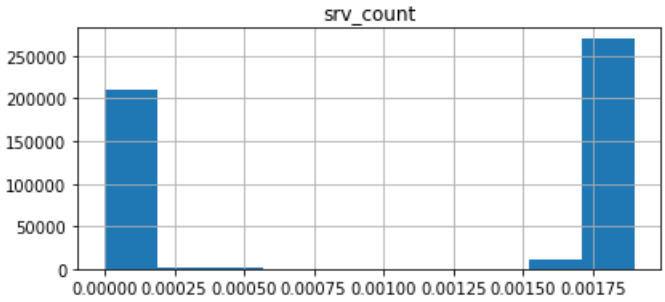

num_file_creations
500000
400000
300000
200000
100000
0
0.0
0.1
0.2
0.3
0.4

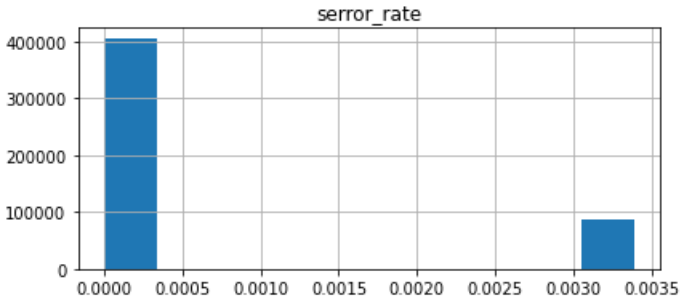

num_shells
500000
400000
300000
200000
100000
0
0.00
0.05
0.10
0.15
0.20
0.25

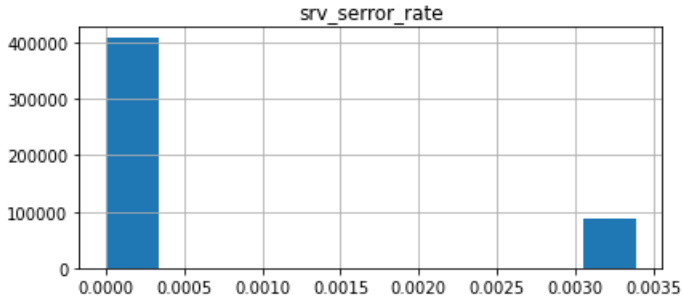

num_access_files
500000
400000
300000
200000
100000
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30

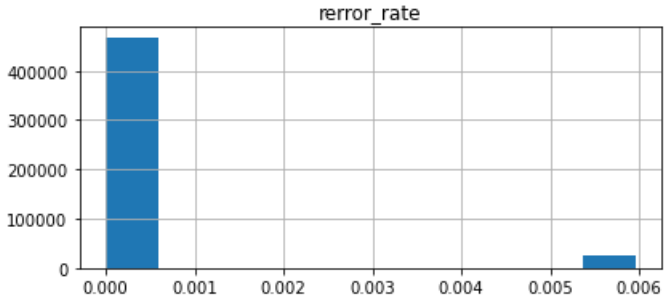

num_outbound_cmds
500000
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4


count
250000
200000
150000
100000
50000
0
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175

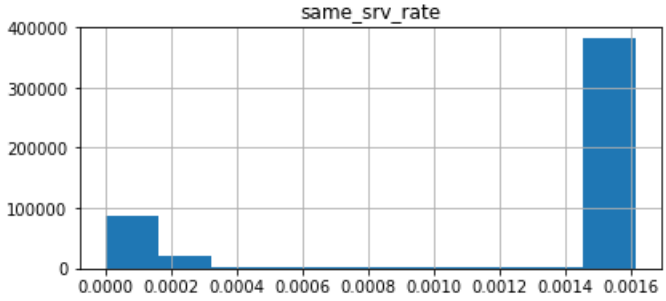

srv_count
250000
200000
150000
100000
50000
0
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175

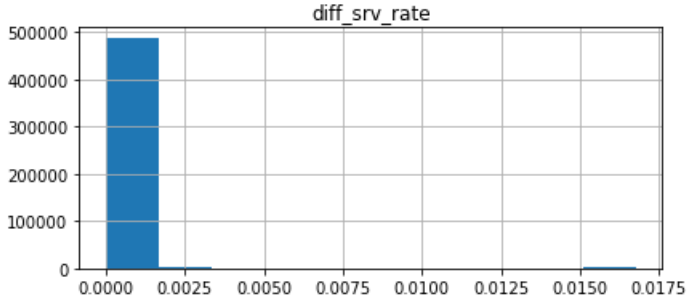

serror_rate
400000
300000
200000
100000
0
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035

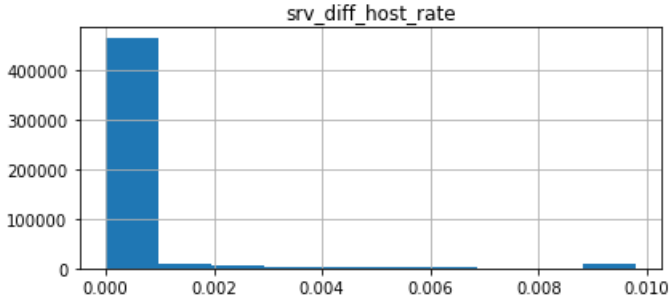

srv_serror_rate
400000
300000
200000
100000
0
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035

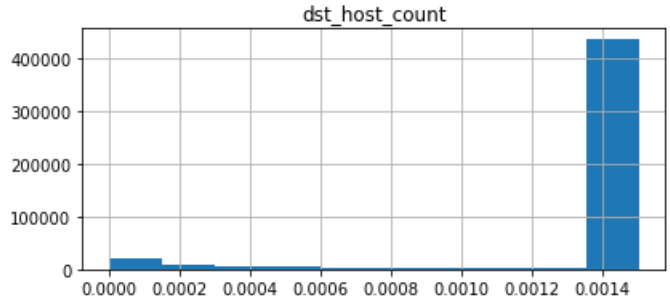

rerror_rate
400000
300000
200000
100000
0
0.000
0.001
0.002
0.003
0.004
0.005
0.006

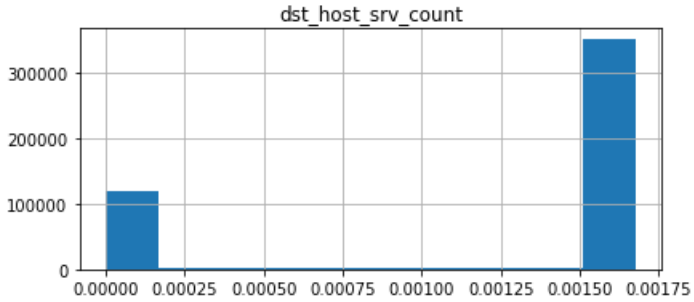

srv_rerror_rate
400000
300000
200000
100000
0
0.000
0.001
0.002
0.003
0.004
0.005
0.006

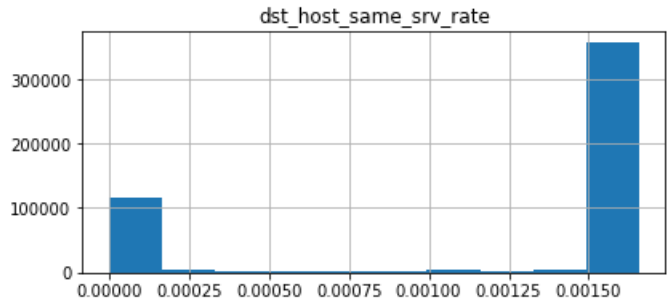

same_srv_rate
400000
300000
200000
100000
0
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
0.0014
0.0016

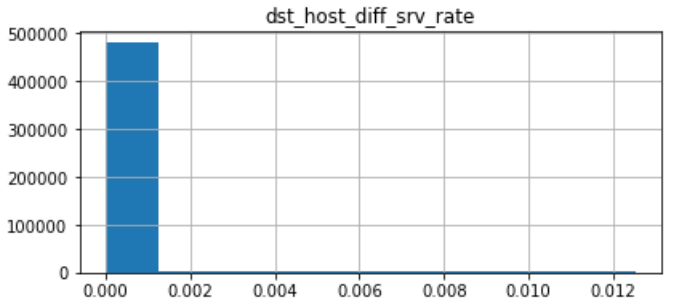

diff_srv_rate
500000
400000
300000
200000
100000
0
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175

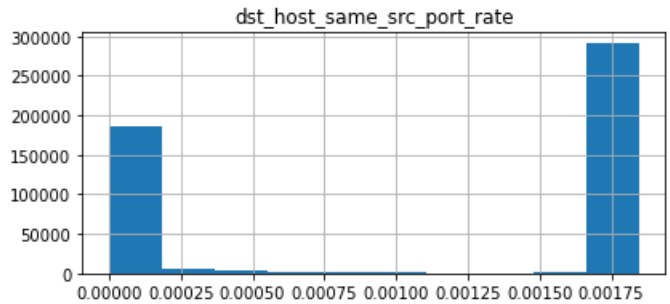

srv_diff_host_rate
400000
300000
200000
100000
0
0.000
0.002
0.004
0.006
0.008
0.010

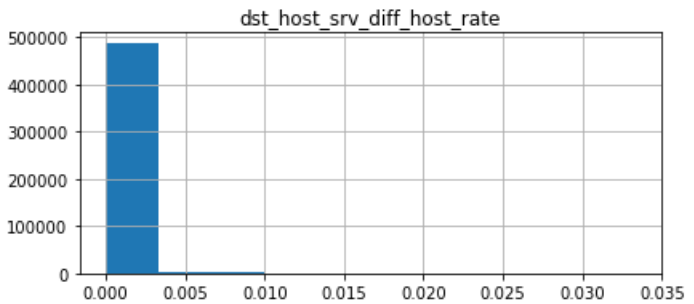

dst_host_count
400000
300000
200000
100000
0
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
0.0014

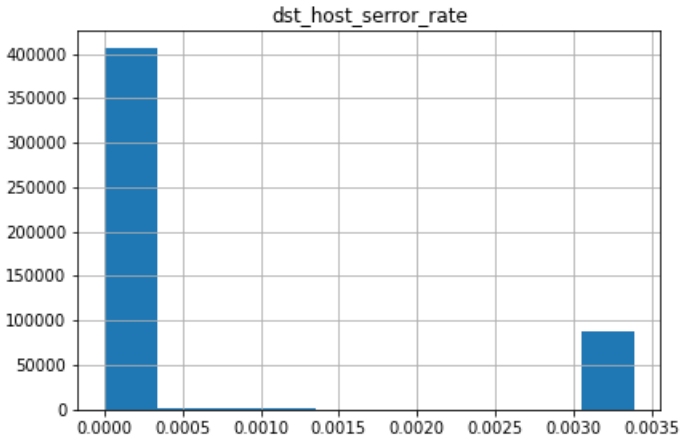

dst_host_srv_count
300000
200000
100000
0
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175

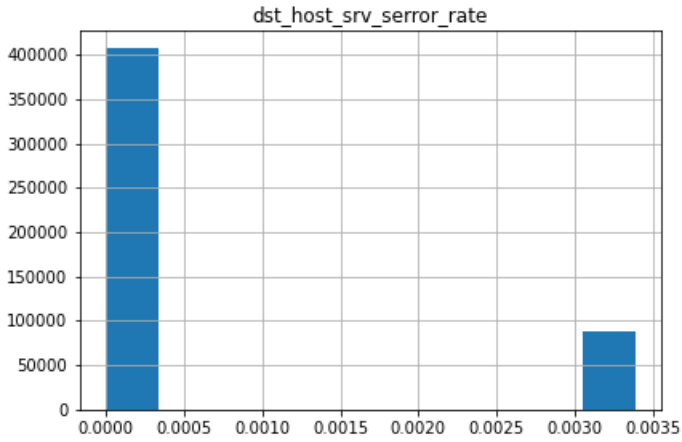

dst_host_same_srv_rate
300000
200000
100000
0
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150

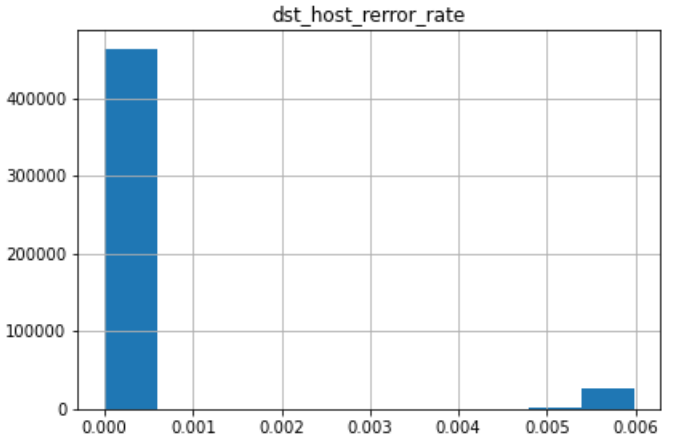

dst_host_diff_srv_rate
500000
400000
300000
200000
100000
0
0.000
0.002
0.004
0.006
0.008
0.010
0.012

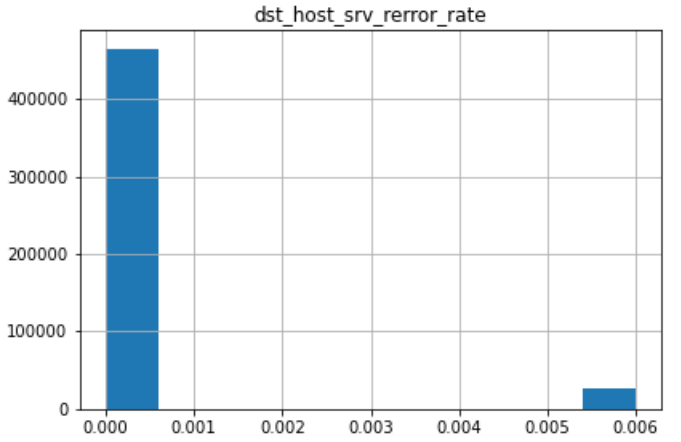

dst_host_same_src_port_rate
300000
250000
200000
150000
100000
50000
0
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175


dst_host_srv_diff_host_rate
500000
400000
300000
200000
100000
0
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035









4 Modeling

4.1 Model that over-fits the entire dataset

To understand the dataset and to estimate the model size (architecture and hyper-parameters) , we decided to build a model that over-fits the entire dataset. This is an experiment step to understand what model size would be required to overfit entire data. We performed below steps:

- Step-1: Using ENTIRE dataset to "OVERFIT" the model using vannila (single neuron) logistic regression model to reach accuracy 100 percent or close to 100 percent

- Step-2: If the accuracy did not reach 100 percent, then a bigger architecture model will be designed and modeled to achieve accuracy of 100 percent or close to 100 percent.

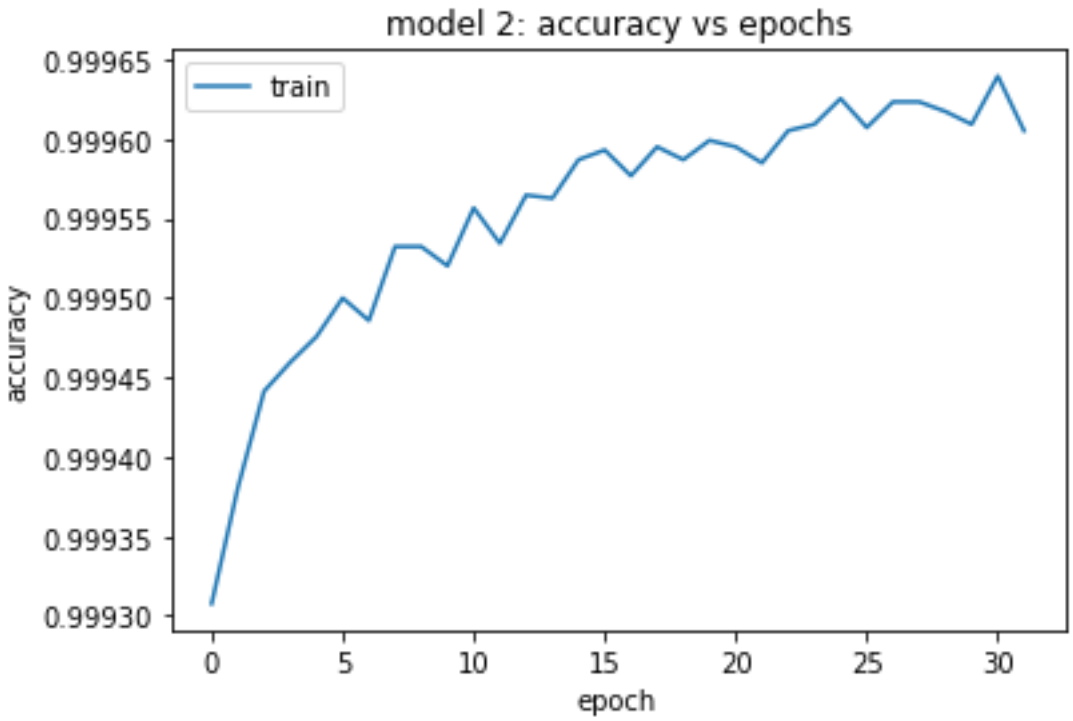
4.1.1 Model-1 Architecture Summary Diagram

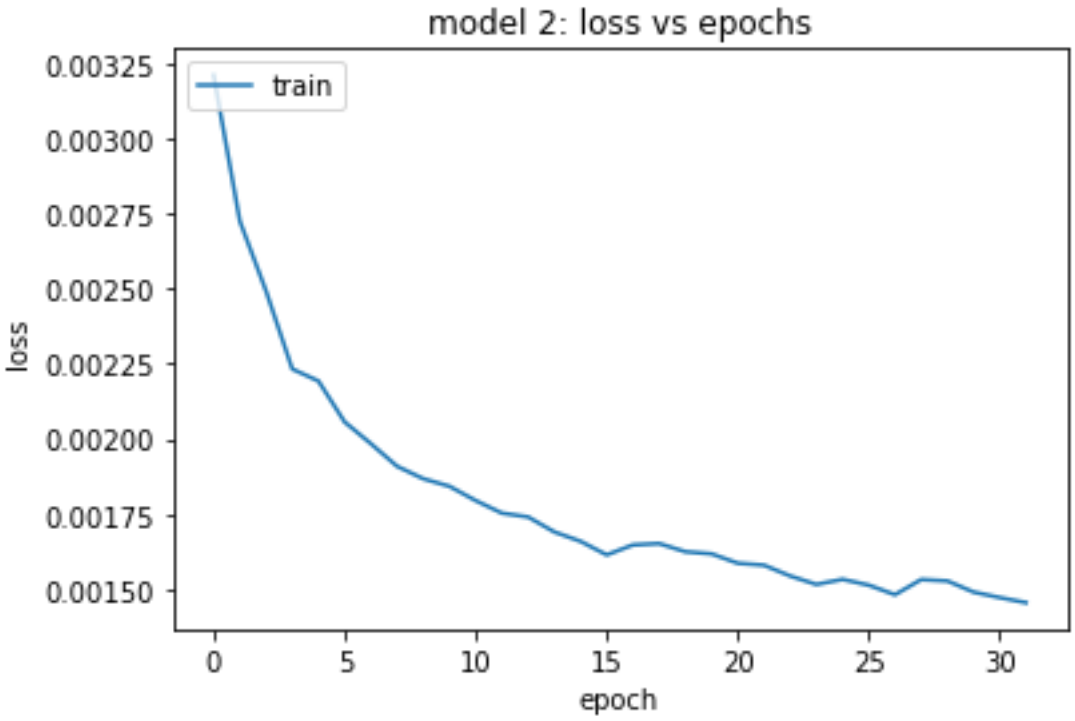
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 1)                 120
=================================================================
Total params: 120
Trainable params: 120
Non-trainable params: 0
_________________________________________________________________
```

4.1.2 Model-2: Architecture Summary Diagram

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 8)                 960
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 36
_________________________________________________________________
dense_13 (Dense)             (None, 2)                 10
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 3
=================================================================
Total params: 1,009
Trainable params: 1,009
Non-trainable params: 0
```

4.1.3 Model Performance





4.1.4 Inference

As seen from the above model, the basic logistic regression model has achieved an accuracy of 99.81 with 256 epochs.A seen from the model-2 performance we have achieved 99.96 percent accuracy (close to 100 percent) which means this model did over-fit the dataset. Therefore, a 4-layer neural network model architecture with 8 neurons, 4 neurons and 2 neurons in internal layers is sufficient enough to overfit the entire dataset.

4.2 Generalized Model

- Shuffling - Train-Validation Split - Comparision of different algorithms

5 Conclusion

Will be updated in next phase

References

[1] Manoj Kumar Putchala. Deep learning approach for intrusion detection system (ids) in the internet of things (iot) network using gated recurrent neural networks (gru). 2017.

[2] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[3] Steven Huang. Kdd cup 1999 data, computer network intrusion detection (version 1). 2018. Available from https://www.kaggle.com/galaxyh/kdd-cup-1999-data.