Network Anomaly Detection Report

Charles Suresh

09/08/2020

Contents

1 Overview

This Report is related to the Capstone Project under the HarvardX Data Science Course: PH125.9x. With the increasing reliance on technology, it is becoming more and more essential to secure every aspect of online information and data. As the internet grows and computer networks become bigger, network security has become one of the most important aspects for organizations to consider. While there is no network that is immune to attacks, a stable and efficient network security system is essential to protecting data and for seamless operations.

To goal of this project was to build a network intrusion detection system that can do two things:

1. Detect whether a network acitivity is normal or is an attack (Binomial Classification)
2. To classify the type of network attack as:

    a) Normal
    b) DoS (Denial of Service)
    c) Probe
    d) R2L (Remote to Local/User)
    e) U2R (User to Root)

The dataset used for building the network intrusion detection system is from kaggle: https://www.kaggle.com/anushonkar/network-anamoly-detection

The available dataset is already split into Train and Test sets. However, the two sets were combined and split into 'edx' set and 'validation' set using the'createDataPartition' function to ensure even distrubition of different attack types. Algorithms were developed and tested by further splitting the 'edx' set into 'Train' and 'Test' sets. After selecting a suitable model, the entire 'edx' set was used to train the algorithm and make the final predictions on the 'validation' set. The metric used for assessing the models are:

1. For the Binomail Classifiction: Accuracy, Sensitivity and Specificity
2. For the Multinomial Classification: Accuracy

Because of the large number of features in the dataset, dimension reduction was attempted. Since substantial dimension reduction could not be achieved, Random Forest and Decision Trees, on account of being the most suited to datasets with high dimensions, were used.

2 Data Preparation

In this stafe, the Train and Test sets are combined and the following changes are made it:

1. For the first goal, a new column: "detect" is created with either of two entires: 'normal' or 'abnormal'. All attacks other than 'normal' will be categorized as 'abnormal'
2. For the second goal, which is to classify the type of attack, a new column "attack_class" is created, which takes in one of 5 values: a) DoS b) Probe c) R2L d) U2R or e) normal. The entry in the "attack_class" column will be dependent on the attack name in the "attack" column:

a) DoS attack class encomposses the following attack types: back, land, neptune, pod, smurf, teardrop, apache2, udpstorm, processtable and worm.

b) Probe attack class encomposses the following attack types: satan, ipsweep, nmap, portsweep, mscan, saint

c) R2L attack class encomposses the following attack types: guess_passwd, ftp_write, imap, phf, multihop, warezmaster, warezclient, spy, xloc, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named

d) U2R attack class encomposses the following attack types: buffer_overflow, loadmodule, rootkit, perl, sqlattack, xterm, ps

After these changes to the dataset, the entire set is split into 'edx_set' and 'validation_set' using the'createDataPartition' function to ensure even distrubition of different attack types in the ratio 8:2

Only the edx_set will be used for training and developing the model.

3 Data Description

A brief decription of each of the attack classes:

a) DoS (Denial of service) is a attack category in which the victim's resources are depleted, thereby making it unable to handle legitimate requests

b) Probe: The objective of the attacks under this class is to gain information about the remote victim e.g. port scanning

c) R2L: In unauthorized access from a remote machine (R2L), the attacker intrudes into a remote machine and gains local access of the victim machine.

d) U2R (User to Root): In the attacks under this class, an attacker uses a normal account to login into a victim system and tries to gain root/administrator privileges by exploiting some vulnerability in the victim

The features that can be used for making predictions and their descriptions:

1. Duration: Length of time duration of the connection
2. Protocol_type: Protocol used in the connection (TCP, UPD or ICMP)
3. Service: Destination network service used
4. Flag: Status of the connection. Different flags:

```
S0: Connection attempt seen, no reply

S1: Connection established, not terminated

S2: Connection established, initiator has closed their side

S3: Connection established, responder has closed their side

SF: Normal establishment and termination. Note that this is the same symbol as for state S1.
The two differ by the number of Byte Counts in the summary. S1 has no Byte Counts, while SF
has more than one

SH: Originator sent a SYN followed by a FIN. Never received a SYN ACK from the responder
(hence the connection is "half" open).

REJ: Connection attempt rejected

RSTO: Connection established, originator aborted (sent a RST)

RSTR: Connection established, responder aborted

RSTOS0: Originator sent a SYN followed by a   RST. Didnt receive a SYN-ACK from   the responder
```

5. Src_bytes: Number of data bytes transferred from source to destination in a single connection
6. Dst_bytes: Number of data bytes transferred from destination to source in a single connection
7. Land: If source and destination IP addresses and port numbers are equal then, this variable takes value 1, else 0
8. Wrong_fragment: Total number of wrong fragments in this connection
9. Urgent: Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated

CONTENT RELATED FEATURES OF EACH NETWORK CONNECTION VECTOR

10. Hot: Number of "hot indicators in the content such as: entering a system directory, creating programs and executing programs

11. Num_failed _logins: Count of failed login attempts
12. Logged_in Login Status: 1 if successfully logged in; 0 otherwise
13. Num_compromised: Number of “compromised” conditions
14. Root_shell: 1 if root shell is obtained; 0 otherwise
15. Su_attempted: 1 if “su root” command attempted or used; 0 otherwise
16. Num_root: Number of “root” accesses or number of operations performed as a root in the connection
17. Num_file_creations: Number of file creation operations in the connection
18. Num_shells: Number of shell prompts
19. Num_access_files: Number of operations on access control files
20. Num_outbound_cmds: Number of outbound commands in an ftp session
21. Is_hot_login: 1 if the login belongs to the “hot” list i.e. if root or admin; else 0
22. Is_guest_login: 1 if the login is a “guest” login; 0 otherwise

TIME RELATED TRAFFIC FEATURES OF EACH NETWORK CONNECTION VECTOR

23. Count: Number of connections to the same destination host as the current connection in the past two seconds
24. Srv_count: Number of connections to the same service (port number) as the current connection in the past two seconds
25. Serror_rate: The percentage of connections that have SYN errors among the connections aggregated in count
26. Srv_serror_rate: The percentage of connections that have SYN errors among the connections aggregated in srv_count
27. Rerror_rate: The percentage of connections that have REJ errors among the connections aggregated in count
28. Srv_rerror_rate: The percentage of connections that have REJ errors among the connections aggregated in srv_count
29. Same_srv_rate: The percentage of connections to the same service, among the connections aggregated in count
30. Diff_srv_rate: The percentage of connections to different services, among the connections aggregated in count
31. Srv_diff_host_rate: The percentage of connections to different destination machines among the connections aggregated in srv_count

HOST BASED TRAFFIC FEATURES

32. Dst_host_count: Number of connections that have the same destination host IP address
33. Dst_host_srv_count: Number of connections that have the same port number
34. Dst_host_same_srv_rate: The percentage of connections to the same service, among the connections aggregated in dst_host_count
35. Dst_host_diff_srv_rate: The percentage of connections to different services, among the connections aggregated in dst_host_count
36. Dst_host_same_src_port_rate: The percentage of connections to the same source port, among the connections aggregated in dst_host_srv_count
37. Dst_host_srv_diff_host_rate: The percentage of connections to different destination machines, among the connections aggregated in dst_host_srv_count
38. Dst_host_serror_rate: The percentage of connections that have SYN errors among the connections aggregated in dst_host_count
39. Dst_host_srv_serror_rate: The percent of connections that have SYN errors among the connections aggregated in dst_host_srv_count
40. Dst_host_rerror_rate: The percentage of connections that have REJ errors among the connections aggregated in dst_host_count
41. Dst_host_srv_rerror_rate: The percentage of connections that have REJ errors among the connections aggregated in dst_host_srv_count

4 Data Cleaning

The data given in the edx dataset is already in tidy format (as seen below):

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot
0	tcp	ftp_data	SF	491	0	0	0	0	0
0	udp	other	SF	146	0	0	0	0	0
0	tcp	private	S0	0	0	0	0	0	0
0	tcp	http	SF	232	8153	0	0	0	0
0	tcp	http	SF	199	420	0	0	0	0
0	tcp	private	REJ	0	0	0	0	0	0

num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	num_file_creations
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

num_shells	num_access_files	num_outbound_cmds	is_hot_login	is_guest_login	count
0	0	0	0	0	2
0	0	0	0	0	13
0	0	0	0	0	123
0	0	0	0	0	5
0	0	0	0	0	30

srv_count	serror_rate	srv_serror_rate	rerror_rate	srv_rerror_rate	same_srv_rate	diff_srv_rate
2	0.0	0.0	0	0	1.00	0.00
1	0.0	0.0	0	0	0.08	0.15
6	1.0	1.0	0	0	0.05	0.07
5	0.2	0.2	0	0	1.00	0.00
32	0.0	0.0	0	0	1.00	0.00

srv_diff_host_rate	dst_host_count	dst_host_srv_count	dst_host_same_srv_rate
0.00	150	25	0.17
0.00	255	1	0.00
0.00	255	26	0.10
0.00	30	255	1.00
0.09	255	255	1.00

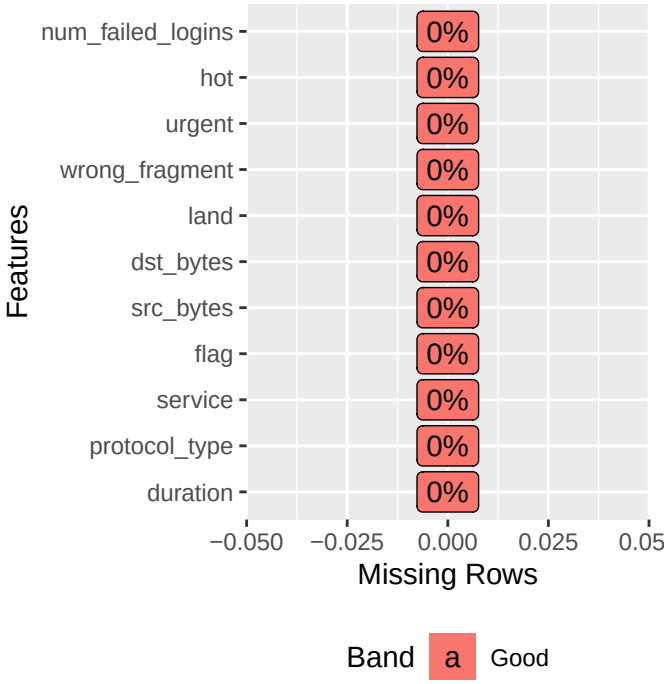
dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate
0.03	0.17	0.00	0.00
0.60	0.88	0.00	0.00
0.05	0.00	0.00	1.00
0.00	0.03	0.04	0.03
0.00	0.00	0.00	0.00

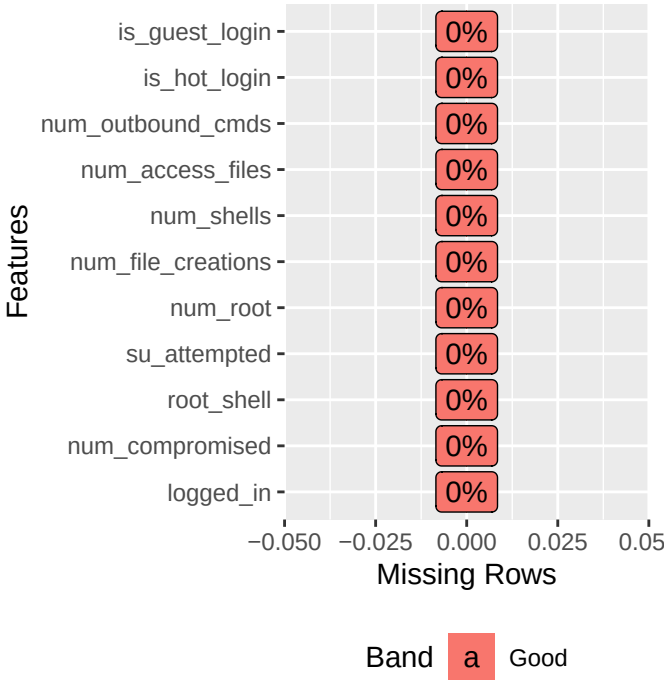
dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	attack	detect	attack_class
0.00	0.05	0.00	normal	normal	normal
0.00	0.00	0.00	normal	normal	normal
1.00	0.00	0.00	neptune	abnormal	dos
0.01	0.00	0.01	normal	normal	normal
0.00	0.00	0.00	normal	normal	normal

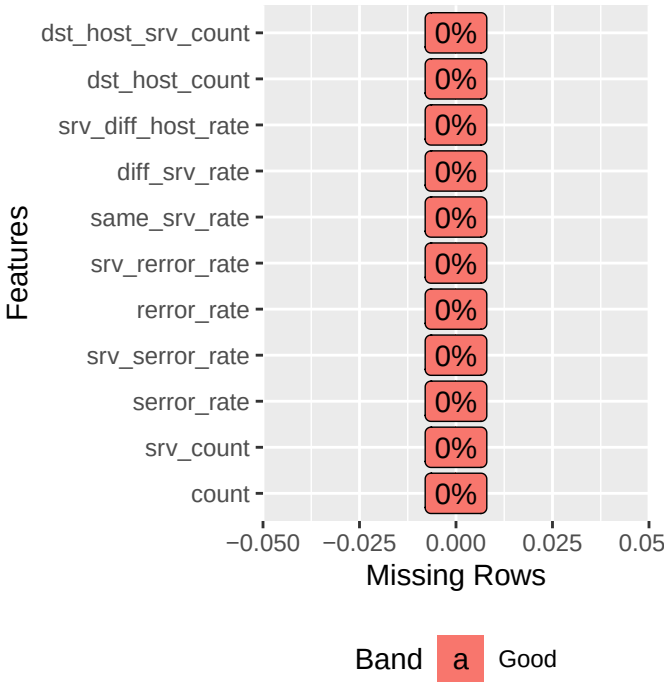
The dimensions of the dataset:

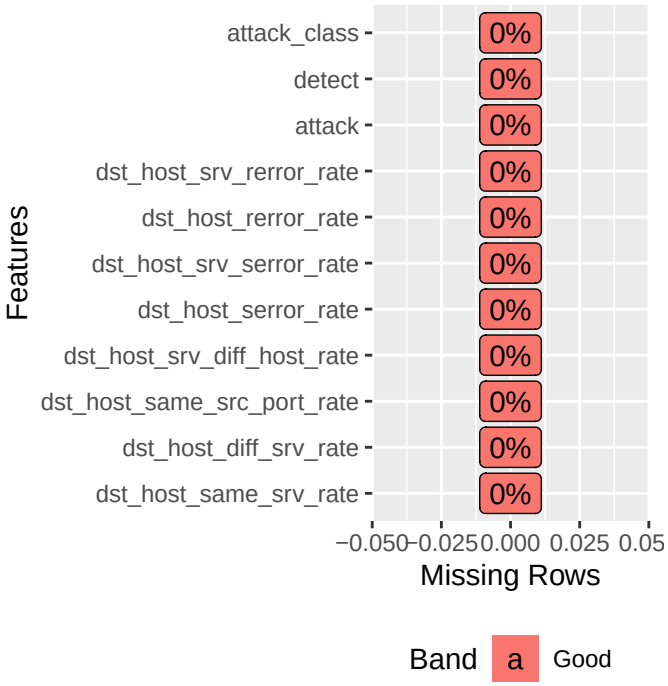
```
##    Rows Columns
##  133664      44
```

From the plots below, we can see that there are no missing values in any of the columns or rows:









5 Data Analysis and Visualisation

Using the summary function to quantitaveley describe all the features in the dataset:

```
##     duration      protocol_type     service           flag
##  Min.   :    0   tcp :109369   http    :43355   SF     :80848
##  1st Qu.:    0   udp : 15876   private :23940   S0     :33173
##  Median :    0   icmp:  8419   domain_u: 8964   REJ    :13583
##  Mean   :  277                 smtp    : 7440   RSTR   : 2758
##  3rd Qu.:    0                 ftp_data: 6932   RSTO   : 2099
##  Max.   :57715                 other   : 4707   S1     :  354
##                                (Other) :38326   (Other):  849
##    src_bytes            dst_bytes          land       wrong_fragment
##  Min.   :         0   Min.   :         0   0:133635   Min.   :0.00
##  1st Qu.:         0   1st Qu.:         0   1:    29   1st Qu.:0.00
##  Median :        44   Median :         0              Median :0.00
##  Mean   :     38534   Mean   :     18646              Mean   :0.02
##  3rd Qu.:       278   3rd Qu.:       566              3rd Qu.:0.00
##  Max.   :1379963888   Max.   :1309937401              Max.   :3.00
##
##      urgent       hot         num_failed_logins logged_in num_compromised
##  Min.   :0   Min.   :  0.0   Min.   :0         0:79849   Min.   :   0
##  1st Qu.:0   1st Qu.:  0.0   1st Qu.:0         1:53815   1st Qu.:   0
##  Median :0   Median :  0.0   Median :0                   Median :   0
##  Mean   :0   Mean   :  0.2   Mean   :0                   Mean   :   0
##  3rd Qu.:0   3rd Qu.:  0.0   3rd Qu.:0                   3rd Qu.:   0
##  Max.   :3   Max.   :101.0   Max.   :5                   Max.   :7479
##
##  root_shell  su_attempted      num_root    num_file_creations   num_shells
##  0:133462   Min.   :0.000   Min.   :   0   Min.   :  0        Min.   :0
##  1:   202   1st Qu.:0.000   1st Qu.:   0   1st Qu.:  0        1st Qu.:0
##             Median :0.000   Median :   0   Median :  0        Median :0
##             Mean   :0.001   Mean   :   0   Mean   :  0        Mean   :0
##             3rd Qu.:0.000   3rd Qu.:   0   3rd Qu.:  0        3rd Qu.:0
##             Max.   :2.000   Max.   :7468   Max.   :100        Max.   :5
##
##  num_access_files num_outbound_cmds is_hot_login is_guest_login     count
##  Min.   :0        Min.   :0         0:133653     0:132031       Min.   :  0
##  1st Qu.:0        1st Qu.:0         1:    11     1:  1633       1st Qu.:  2
##  Median :0        Median :0                                     Median : 13
##  Mean   :0        Mean   :0                                     Mean   : 83
##  3rd Qu.:0        3rd Qu.:0                                     3rd Qu.:141
##  Max.   :9        Max.   :0                                     Max.   :511
##
##    srv_count    serror_rate    srv_serror_rate  rerror_rate    srv_rerror_rate
##  Min.   :  0   Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:  2   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000
##  Median :  7   Median :0.000   Median :0.000   Median :0.000   Median :0.000
##  Mean   : 28   Mean   :0.257   Mean   :0.255   Mean   :0.138   Mean   :0.138
##  3rd Qu.: 17   3rd Qu.:0.860   3rd Qu.:0.910   3rd Qu.:0.000   3rd Qu.:0.000
##  Max.   :511   Max.   :1.000   Max.   :1.000   Max.   :1.000   Max.   :1.000
##
##  same_srv_rate   diff_srv_rate   srv_diff_host_rate dst_host_count
##  Min.   :0.000   Min.   :0.000   Min.   :0.000      Min.   :  0
##  1st Qu.:0.100   1st Qu.:0.000   1st Qu.:0.000      1st Qu.: 87
##  Median :1.000   Median :0.000   Median :0.000      Median :255
##  Mean   :0.673   Mean   :0.068   Mean   :0.098      Mean   :184
##  3rd Qu.:1.000   3rd Qu.:0.060   3rd Qu.:0.000      3rd Qu.:255
##  Max.   :1.000   Max.   :1.000   Max.   :1.000      Max.   :255
##
```

```
##  dst_host_srv_count dst_host_same_srv_rate dst_host_diff_srv_rate
##  Min.   :  0        Min.   :0.000          Min.   :0.000
##  1st Qu.: 11        1st Qu.:0.050          1st Qu.:0.000
##  Median : 72        Median :0.600          Median :0.020
##  Mean   :120        Mean   :0.535          Mean   :0.084
##  3rd Qu.:255        3rd Qu.:1.000          3rd Qu.:0.070
##  Max.   :255        Max.   :1.000          Max.   :1.000
##
##  dst_host_same_src_port_rate dst_host_srv_diff_host_rate dst_host_serror_rate
##  Min.   :0.000               Min.   :0.000              Min.   :0.000
##  1st Qu.:0.000               1st Qu.:0.000              1st Qu.:0.000
##  Median :0.000               Median :0.000              Median :0.000
##  Mean   :0.146               Mean   :0.031              Mean   :0.256
##  3rd Qu.:0.050               3rd Qu.:0.010              3rd Qu.:0.600
##  Max.   :1.000               Max.   :1.000              Max.   :1.000
##
##  dst_host_srv_serror_rate dst_host_rerror_rate dst_host_srv_rerror_rate
##  Min.   :0.000            Min.   :0.000        Min.   :0.000
##  1st Qu.:0.000            1st Qu.:0.000        1st Qu.:0.000
##  Median :0.000            Median :0.000        Median :0.000
##  Mean   :0.251            Mean   :0.136        Mean   :0.136
##  3rd Qu.:0.500            3rd Qu.:0.000        3rd Qu.:0.000
##  Max.   :1.000            Max.   :1.000        Max.   :1.000
##
##        attack           detect      attack_class
##  normal   :69348   normal  :69348   normal:69348
##  neptune  :41274   abnormal:64316   dos   :48048
##  satan    : 3941                    probe :12669
##  ipsweep  : 3365                    r2l   : 3492
##  smurf    : 2989                    u2r   :  107
##  portsweep: 2750
##  (Other)  : 9997
```

It is interesting to note that more that 75% of the entries is '0' in over 16 numerical columns: 'duration', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'num_compromised', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files', 'rerror_rate', 'srv_rerror_rate', 'srv_diff_host_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'

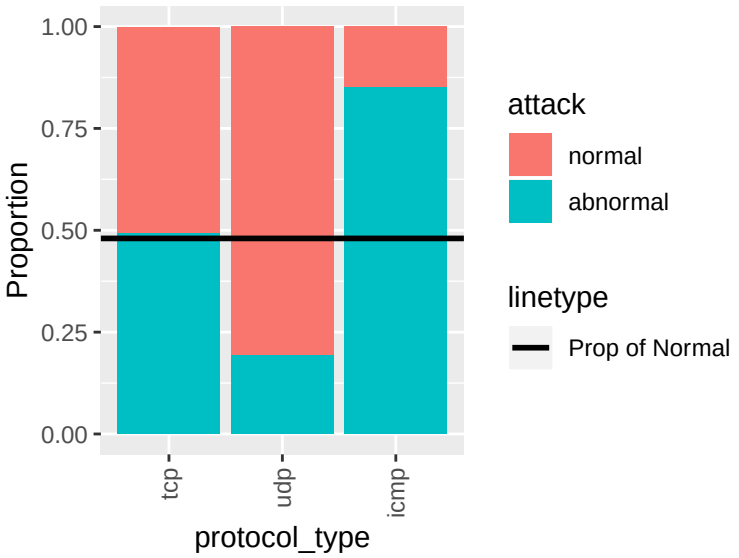
There are also more data points for normal networks than for abnormal ones. Amongst the data points for abnormal networks, DoS attack makes up the highest proportion, far exceeding the rest of the attack classes. Infact, the number of data points for DoS attacks are higher than the rest of the attack classes combined. U2R class has only 107 data points. The attack 'neptune' makes up close to 90% of the DoS type attacks.

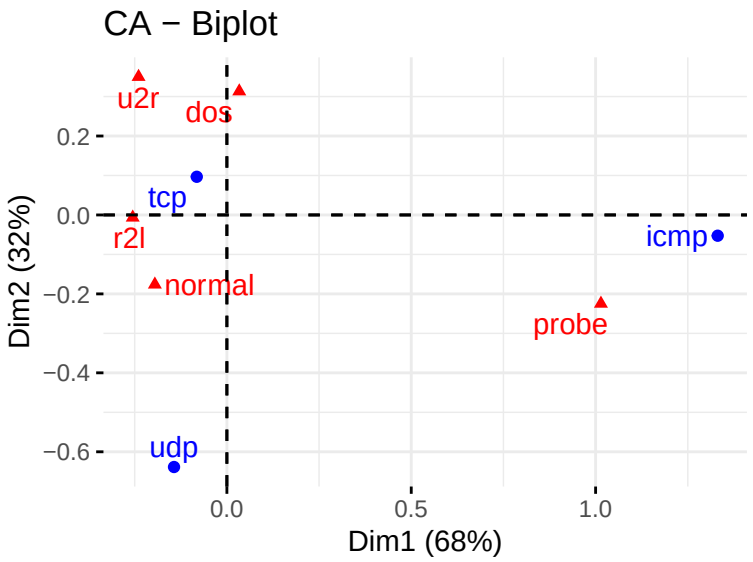
From the summary of the dataset, we can also see that the column: num_outbound_cmds has the number '0' as the value for all entries. Since this value is constant, we can remove this column entirely.

```
edx_set <- edx_set[,-20]
```

5.1 Categorical Data

Contingecy tables and bar plots will be used to perform Data Analaysis and visualisation of Categorical features. For categorical features that can take up more than two categories, the contingency table will be used to perform Correspondance Analysis. The Biplots from the Corresponance Analysis will be used to determine the affinities between the attack types and the categorical feature.

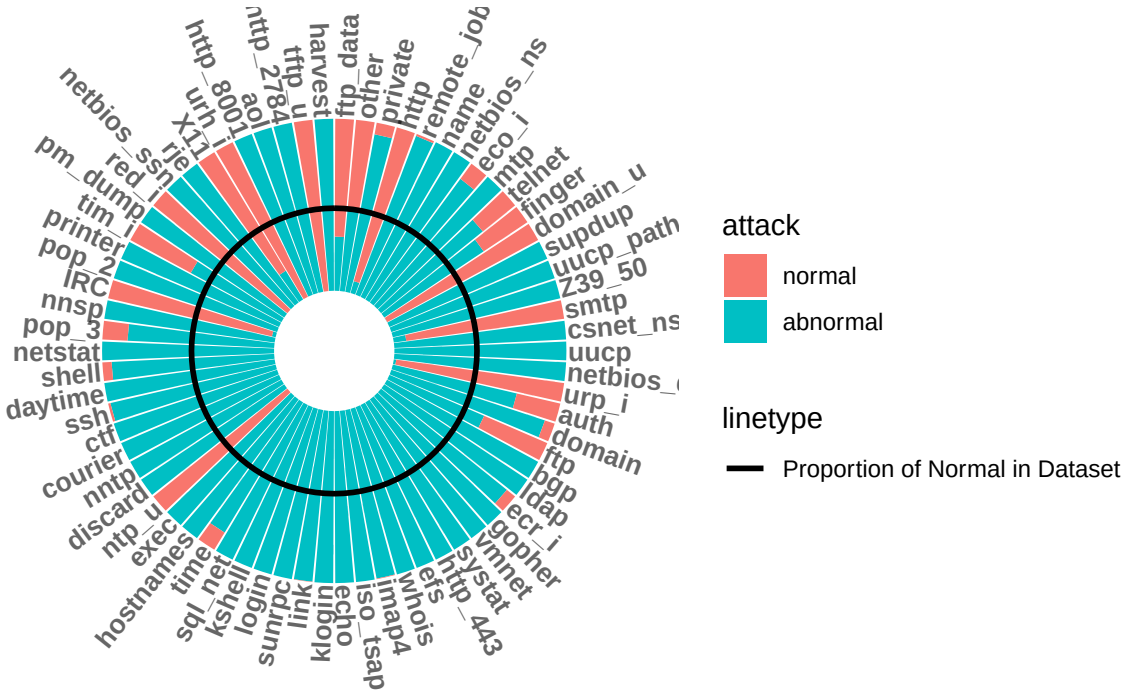


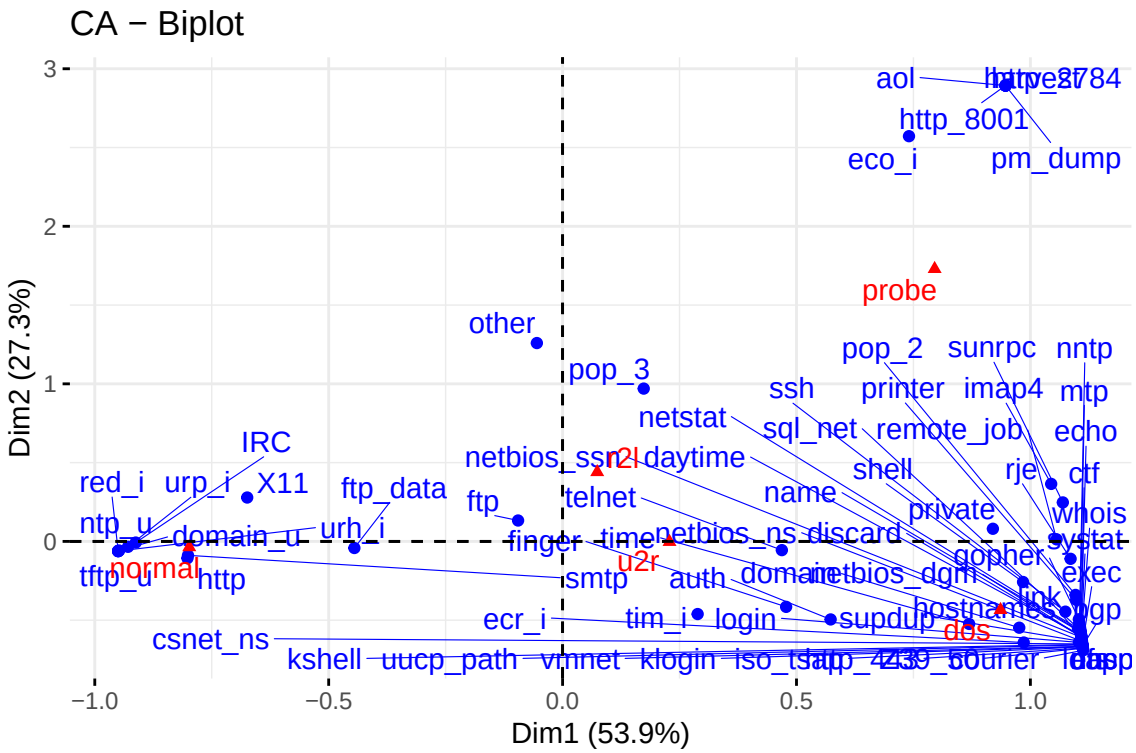


protocol_type	normal	dos	probe	r2l	u2r
tcp	51	40.2	6.3	2.76	0.10
udp	81	5.1	11.2	2.94	0.01
icmp	15	38.1	47.0	0.05	0.00
Tot_Prop	52	36.0	9.5	2.61	0.08

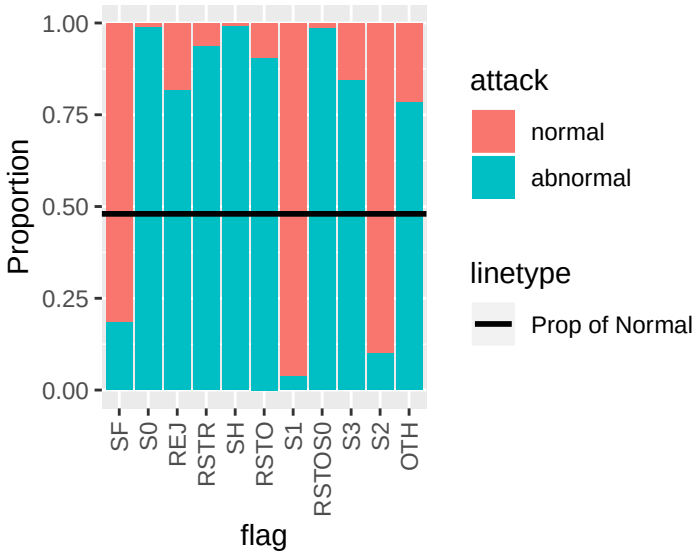
protocol_type	normal	dos	probe	r2l	u2r	Tot_Prop
tcp	79.7	91.6	55	86.54	98.1	81.8
udp	18.5	1.7	14	13.34	1.9	11.9
icmp	1.8	6.7	31	0.11	0.0	6.3

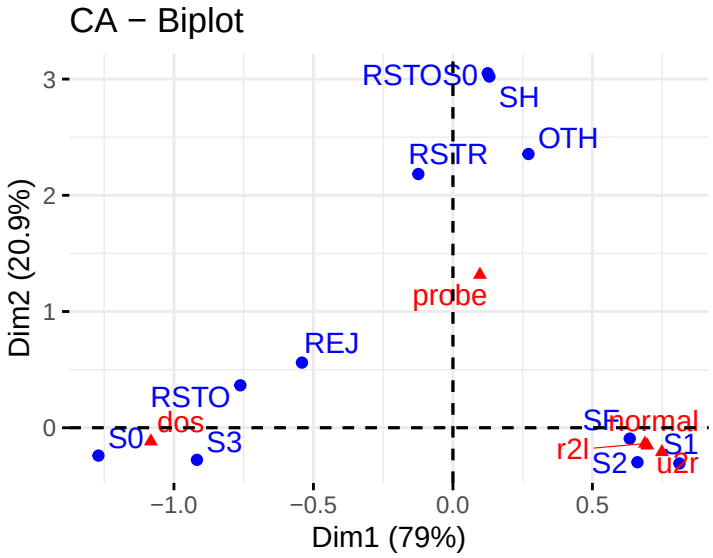
1. Protocol Type: ICMP protocol seems most vulnerable to network attacks with only about 15% of the entries that are normal. It is interesting to note that a far greater proportion (close to 50%) of tcp entries seem to be abnormal as compared to udp (only around 20%), even though theoretically TCP is a bit more reliable than UDP. TCP, on account of being the most reliable, is also the most used protocol. From the Correspondence Analysis Biplot, we can observe that Probe type attack is closely associated to ICMP protocol. We can also see this from the contingency table, where 31% of the Probe attacks where when protocol type was ICMP, even though ICMP protocol accounts for only about 6.3% of the protocols in the entire dataset. We can also observe that U2R and DoS type attacks are most closely associated with TCP. R2L attack is closely associated to TCP, although it also has some association to UDP.



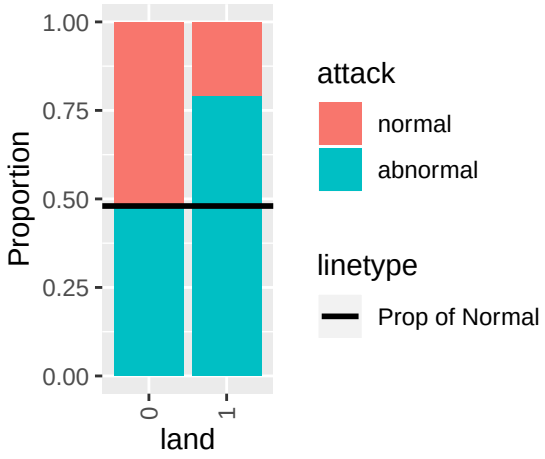


2. Service: From the circular barplot and CA -Biplot, it is clear that service types: tftp_u, urh_i, domain_u, smtp, http, urp_i. ntp_u, IRC, red_i and X11 are almost always normal. Entries with service types: ftp_data and ftp have moderate amounts of abnormalities, with most of them associated with R2L or U2R attack classes. Pop_3 is another service which is almost always associated with either R2L or U2R. Eco_i, aol, pm_dump, http_8001 and harvest service types are almost always related to Probe type attack. Telnet service has close associations with U2R, R2L and DoS attack classes. The rest of the service types are all heavily linked to DoS attack class.





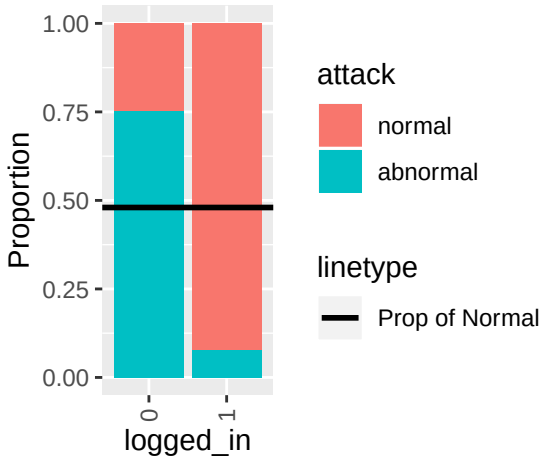
3. Flag: Flag types S0, SH and RSTOS0 are almost always associated with abnormality. DoS type attack class has high associations with flags S0, S3, RSTO and REJ. These flags are all error flags associated with a lack of connection or a sudden termination of connection. Thereby, it makes sense that these flags are all associated with the DoS attack class. SH, RSTOS0 and RSTR are flags typically associated to Probe attack class. In these three flags, a SYN is sent by the originator, however a SYN-ACK is not sent back by the responder (in the case of SH and RSTOS0) or the Responder sends a RST (in the case of RSTR). All these are tell-tale signs of a Probe type attack. SF, S2 and S1 flags are equally associated with Normal, R2L and U2R. These three flags signify a normal connection state value, which means R2L and U2R attack class cannot be differentiated from Normal networks using connection state flags in isolation.



land	normal	dos	probe	r2l	u2r
0	52	36	9.5	2.6	0.08
1	21	79	0.0	0.0	0.00
Tot_Prop	52	36	9.5	2.6	0.08

land	normal	dos	probe	r2l	u2r	Total_Prop
0	99.99	99.95	100	100	100	99.98
1	0.01	0.05	0	0	0	0.02

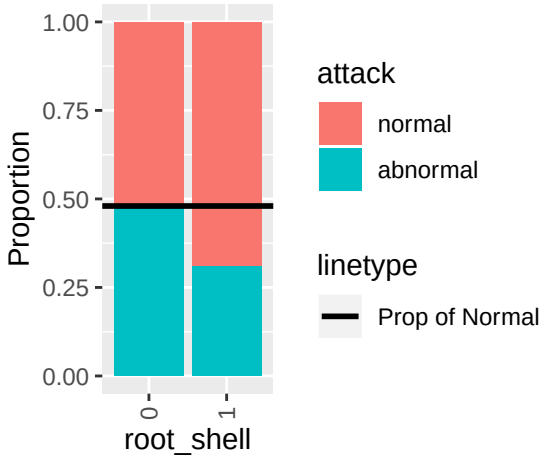
4. Land: When land is 1, i.e. when the connection is from/to the same host/port, around 79% of the time, the network attack class is DoS, even though DoS attack class accounts for only 36% of the network attack entries in the entire dataset. The remaining 21% of the time, the network is normal. Probe, R2L and U2R attacks never seem to occur when Land is 1



logged_in	normal	dos	probe	r2l	u2r
0	25	57.8	15.62	1.9	0.02
1	92	3.5	0.37	3.7	0.17
Tot_Prop	52	36.0	9.48	2.6	0.08

logged_in	normal	dos	probe	r2l	u2r	Total_Prop
0	28	96.1	98.4	43	17	60
1	72	3.9	1.6	57	83	40

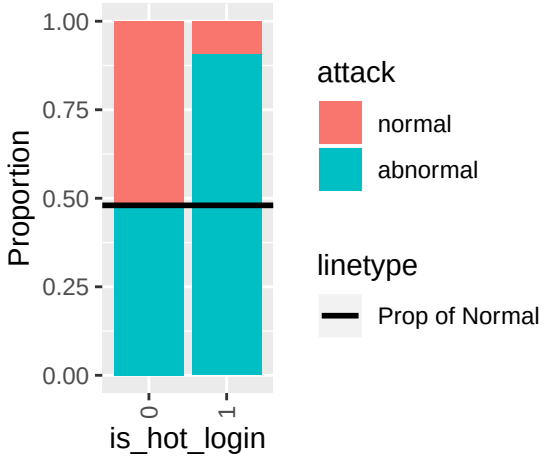
5. Logged In: The chances of a normal network when not logged is low at 28% (even though 60% of the dataset has a normal network). When successfully logged-in i.e. when 'logged_in' is 1, the overall chances of a normal network are higher at 92%. However, the chances of a U2R and R2L attack are considerably higher when logged in as compared to when not logged in. This makes sense since, both U2R and R2L attacks are related to unauthorized access. DoS and Probe attack classes on the other hand hardly ever occur when 'logged_in' is 1 (at 3.9% and 1.6% respectively even though 40% of the dataset has logged_in = 1).



root_shell	normal	dos	probe	r2l	u2r
0	52	36	9.5	2.6	0.04
1	69	0	0.0	6.9	24.26
Tot_Prop	52	36	9.5	2.6	0.08

root_shell	normal	dos	probe	r2l	u2r	Total_Prop
0	99.8	100	100	99.6	54	99.85
1	0.2	0	0	0.4	46	0.15

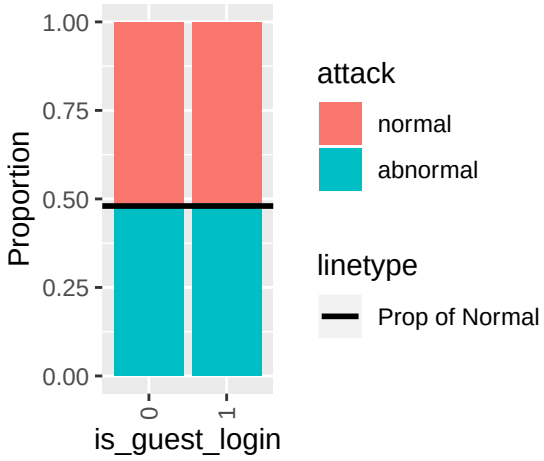
6. Root Shell: The proportion of entries for which root_shell is 1 i.e. root shell access is obtained is quite low (only 0.15%). For entries with root_shell = 1, the percentage of normal entries are higher at 69% as compared to 52% for the entire dataset. However, the proportion of U2R attack also increases exponentially when root_shell is 1. The proportion of R2L attacks increase by over 2.5 times as well, while the proportion of DoS and Probe attacks drop to 0.



is_hot_login	normal	dos	probe	r2l	u2r
0	51.9	36	9.5	2.6	0.08
1	9.1	0	0.0	54.5	36.36
Tot_Prop	51.9	36	9.5	2.6	0.08

is_hot_login	normal	dos	probe	r2l	u2r	Total_Prop
0	100	100	100	99.83	96.3	99.99
1	0	0	0	0.17	3.7	0.01

7. Hot Login: The proportion of entries for which 'is_hot_login' is 1 i.e. the login belongs to hot list is again quite low (only 0.01%). For entries where the login belongs to the hot list, the proportion of abnormalities increase to over 90% when only 48% of the entire dataset are classified as abnormal. The abnormalities that cause this increase are U2R and R2L attack classes that shoot up exponentially as well. This is again because both U2R and R2L attacks are related to unauthorized access. DoS and Probe attacks are at 0, when the login is from a hot list.



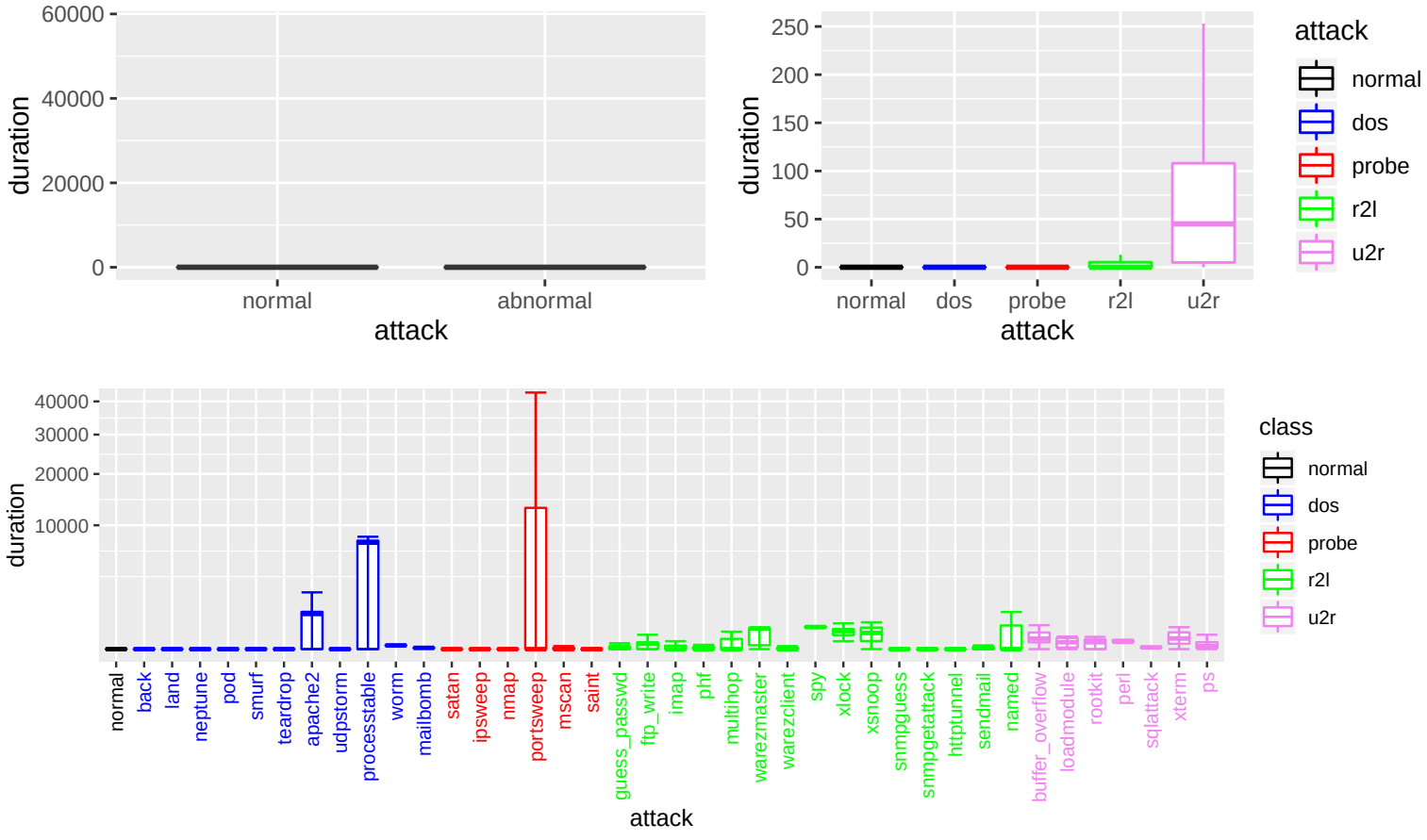
is_guest_login	normal	dos	probe	r2l	u2r
0	52	36	9.59	2.0	0.08
1	51	0	0.06	49.0	0.00
Tot_Prop	52	36	9.48	2.6	0.08

is_guest_login	normal	dos	probe	r2l	u2r	Total_Prop
0	98.8	100	99.99	77	100	98.8
1	1.2	0	0.01	23	0	1.2

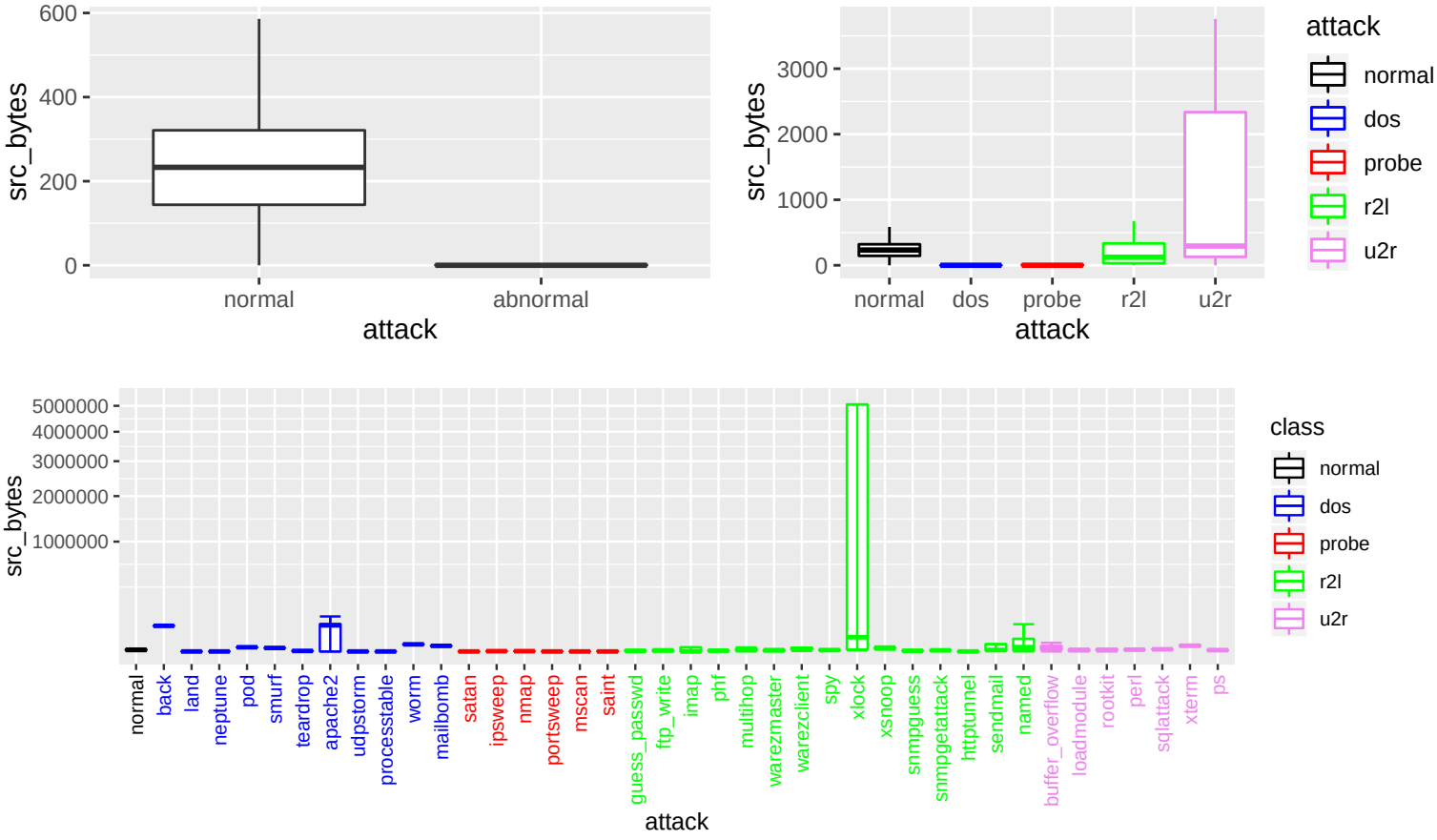
8. Guest Login: When 'is_guest_login' is 1, The proportion of R2L attacks shoot up, while the proportions of DoS, Probe and U2R attacks plunge. The proportion of normal networks remain approximately the same.

5.2 Numerical Data

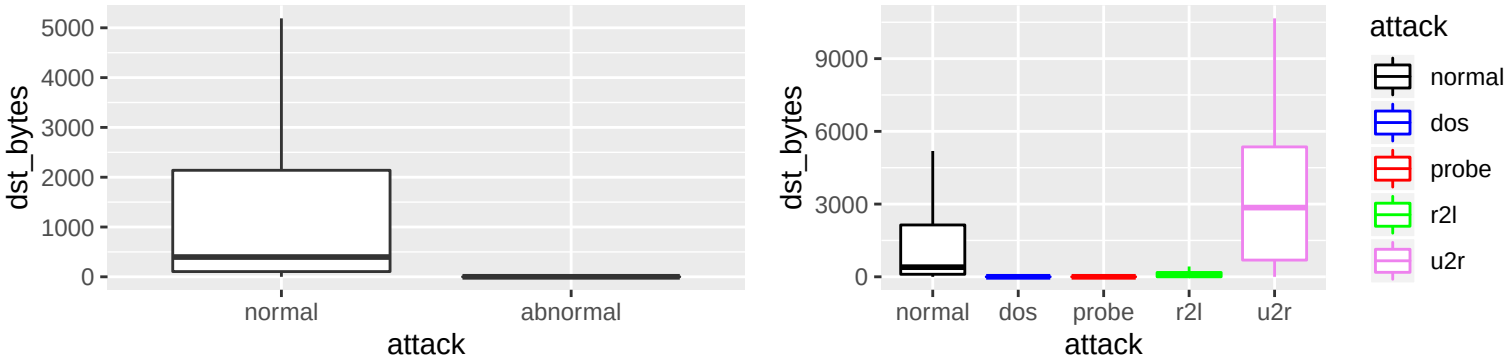
Boxplots will be used to examine the distribution of numerical features for different attack classes and network abnormalities. For some of the plots, a Square Root transormation is done on the y-axis to provide better visual clarity.

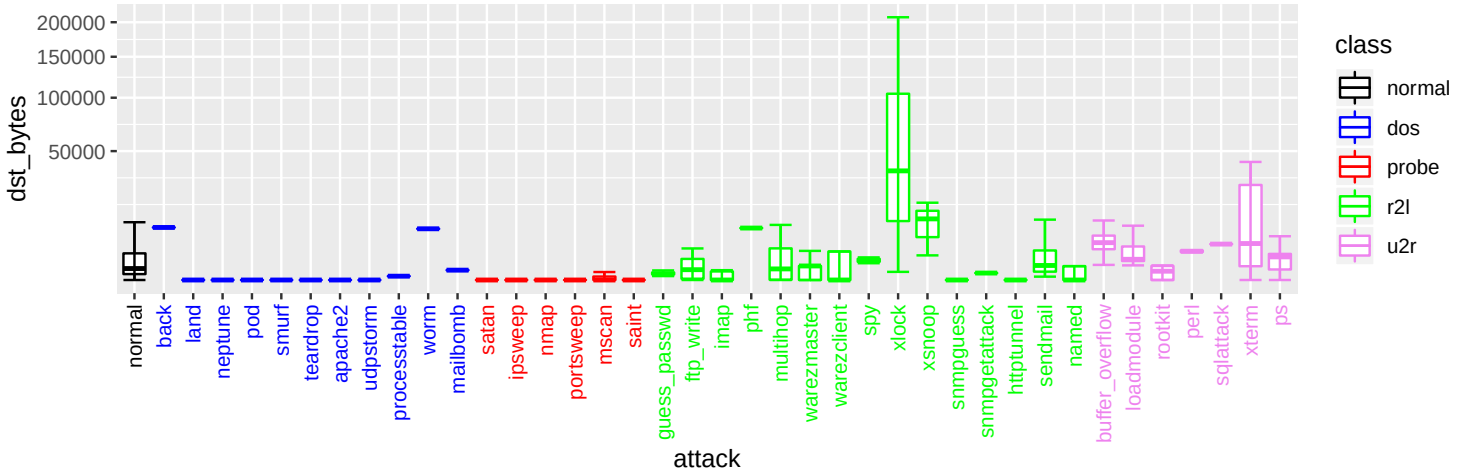


1. Duration: We can see that U2R attack class tends to have a higher duration in general as compared to others. When we further break down the attack classes to specific attacks, we observe that the attack 'processtable' under the DoS attack class seems to have the highest median, followed by 'apache2'. Portsweep attack under the Probe class has a wide distibution.

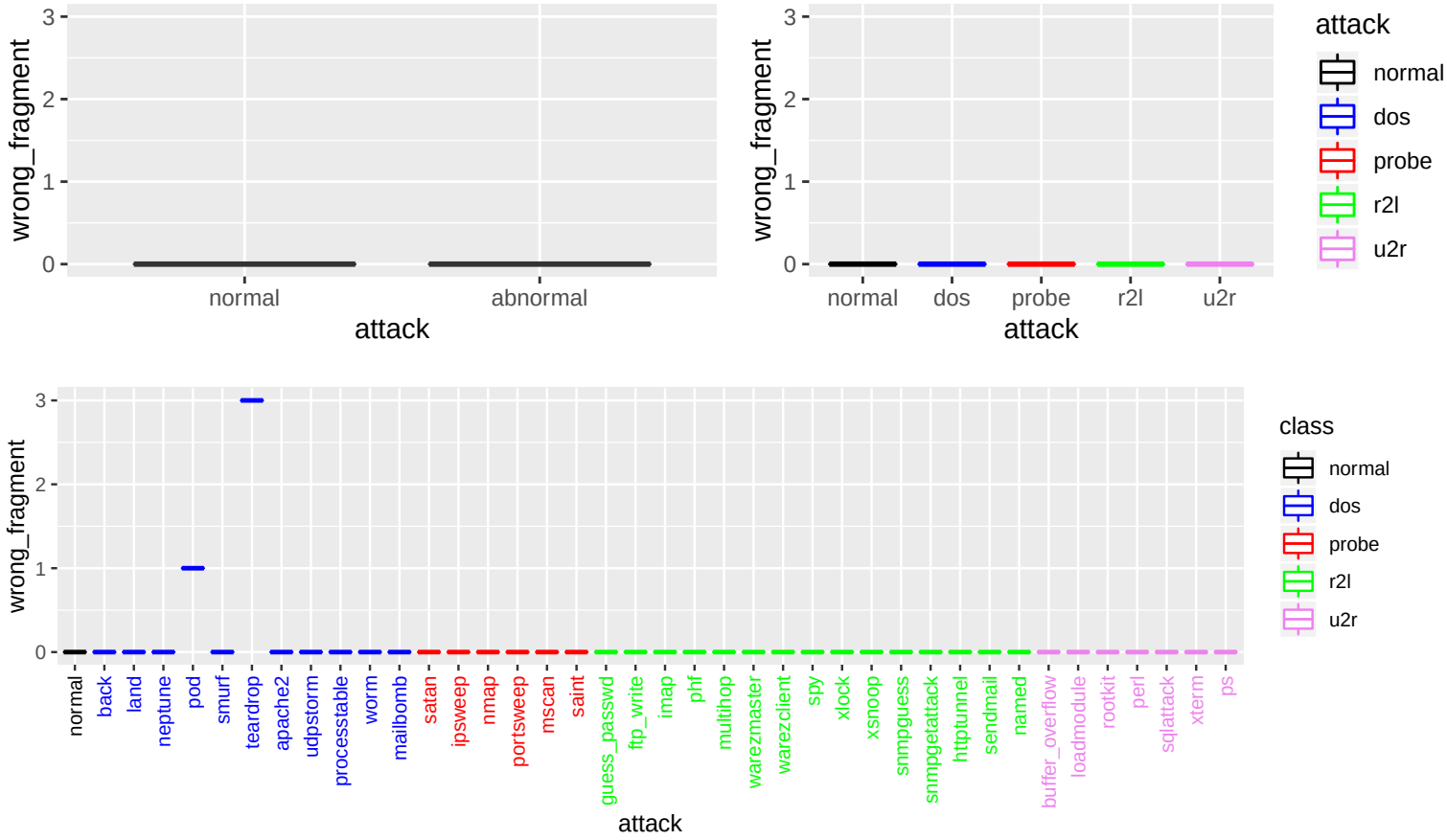


2. Source Bytes: From the boxplots, we can see that normal networks seems to have higher source bytes in general. However, when we break down the abnormalities by attack classes, we observe that the median for U2R attack class is slightly higher than that for normal networks. U2R attacks also have a large Inter-Quartile range, with the 3rd quartile exceeding 2000 bytes. R2L attacks also have higher medians compared to Probe and DoS. This makes sense because, both, R2L and U2R attacks are related to unauthorized access. On further breaking down the attack classes to individual attacks, we that 'xlock' attack under the R2L has a high median with Q3 stretching all the way till 5 million bytes. DoS attacks – 'back' and 'apache2' have the highest medians amongst all other attacks.

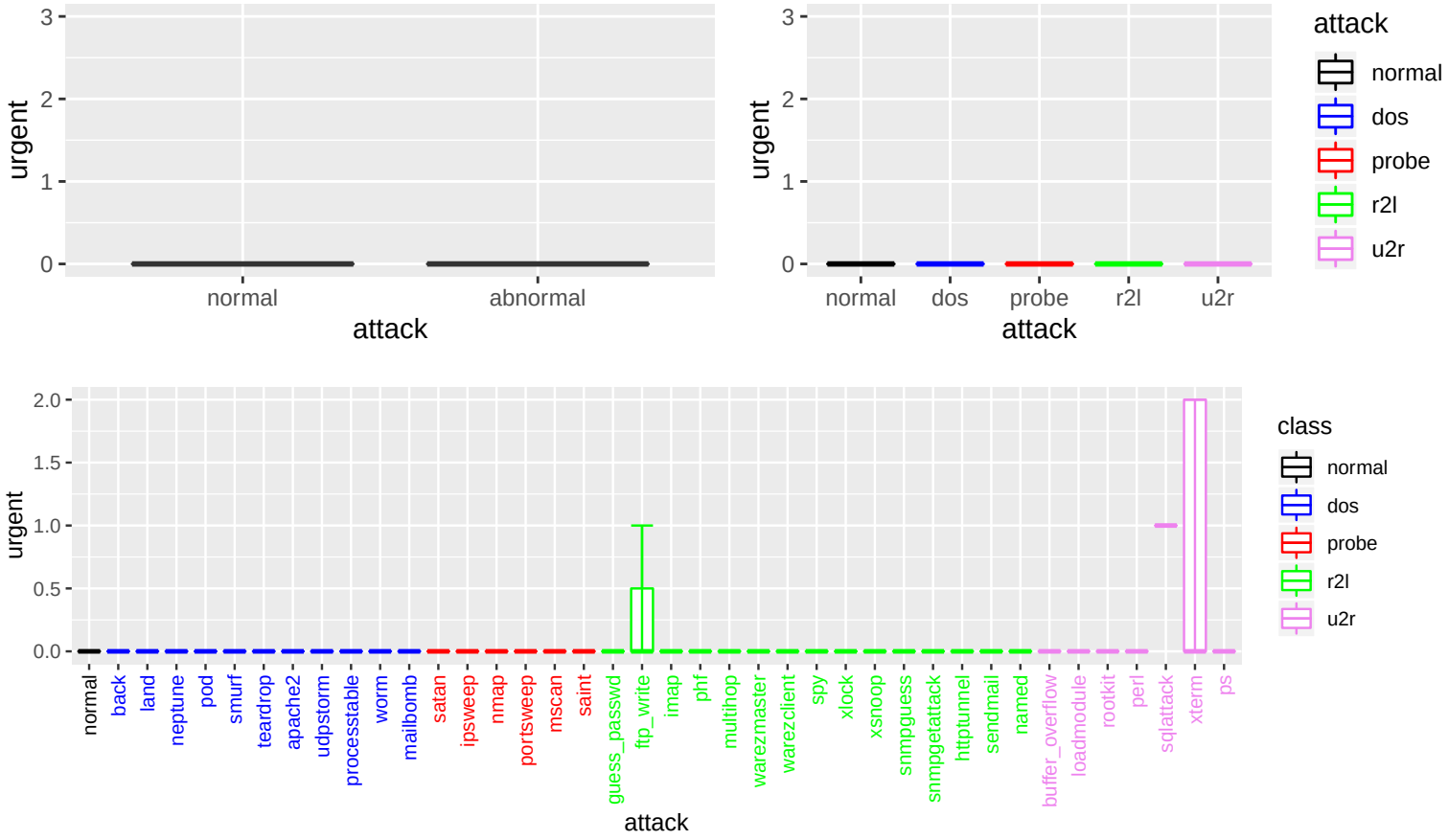




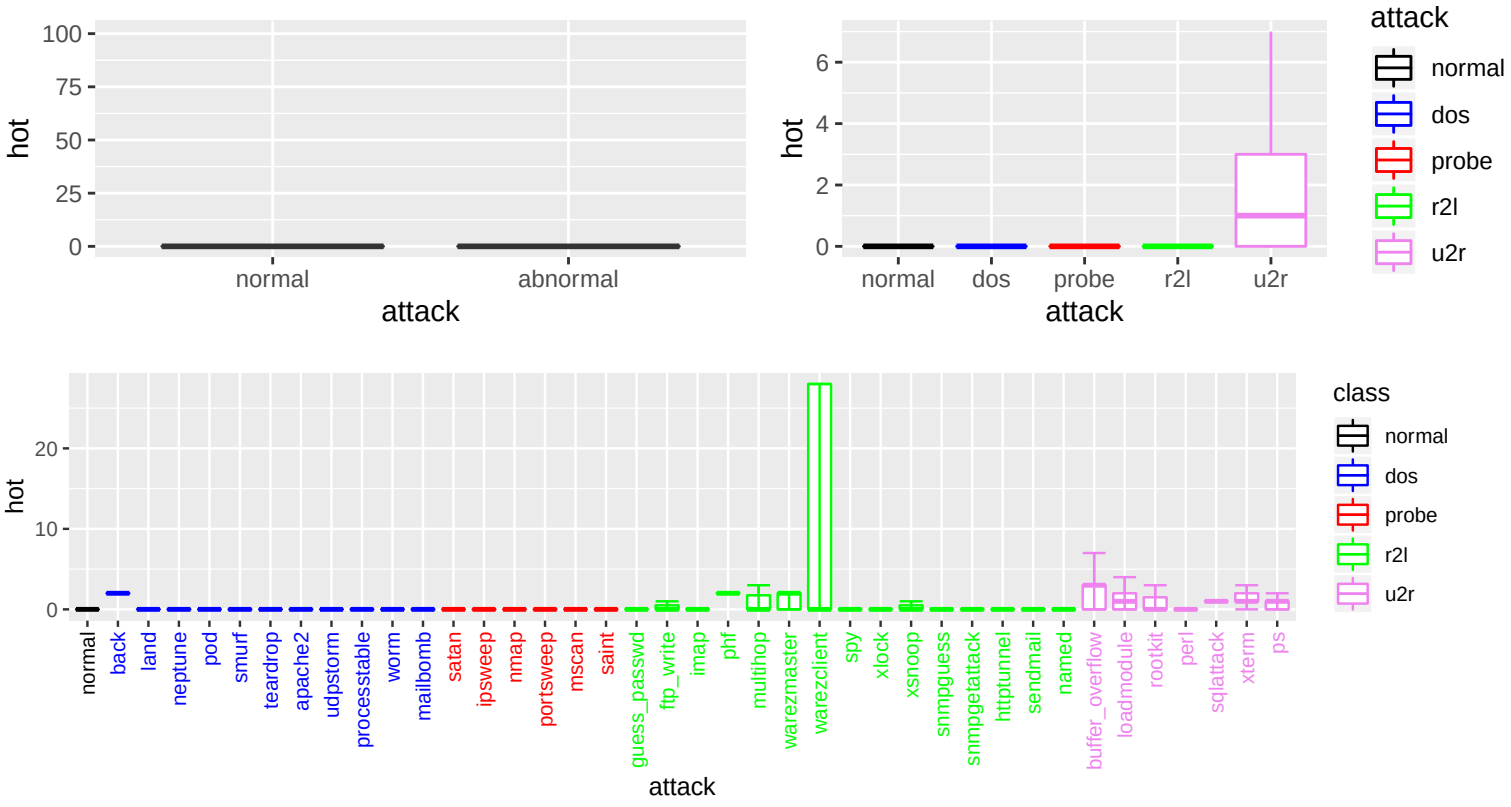
3. Destination Bytes: Normal networks again seem to have higher values of Destination Bytes. When broken down by attack classes, we observe U2R attack class having higher overall values, followed by normal networks and R2L. When we further break down the classes to individual attacks, we can see most of the attack types under the R2L and U2R having high values. This is again because R2L and U2R attacks are related to unauthorized access. Back and Worm attacks under the DoS class also have high values.



4. Wrong Fragment: Pod (Ping of Death) and Teardrop attacks under the DoS attack class are the only attacks that have high values of Wrong Fragments. Teardrop attack involves sending fragmented packets to a target machine. Since the machine receiving such packets cannot reassemble them, the packets overlap one another, crashing the device. In PoD attack, attacker attempts to crash, destabilize, or freeze the targeted computer or service by sending malformed or oversized packets using a simple ping command.

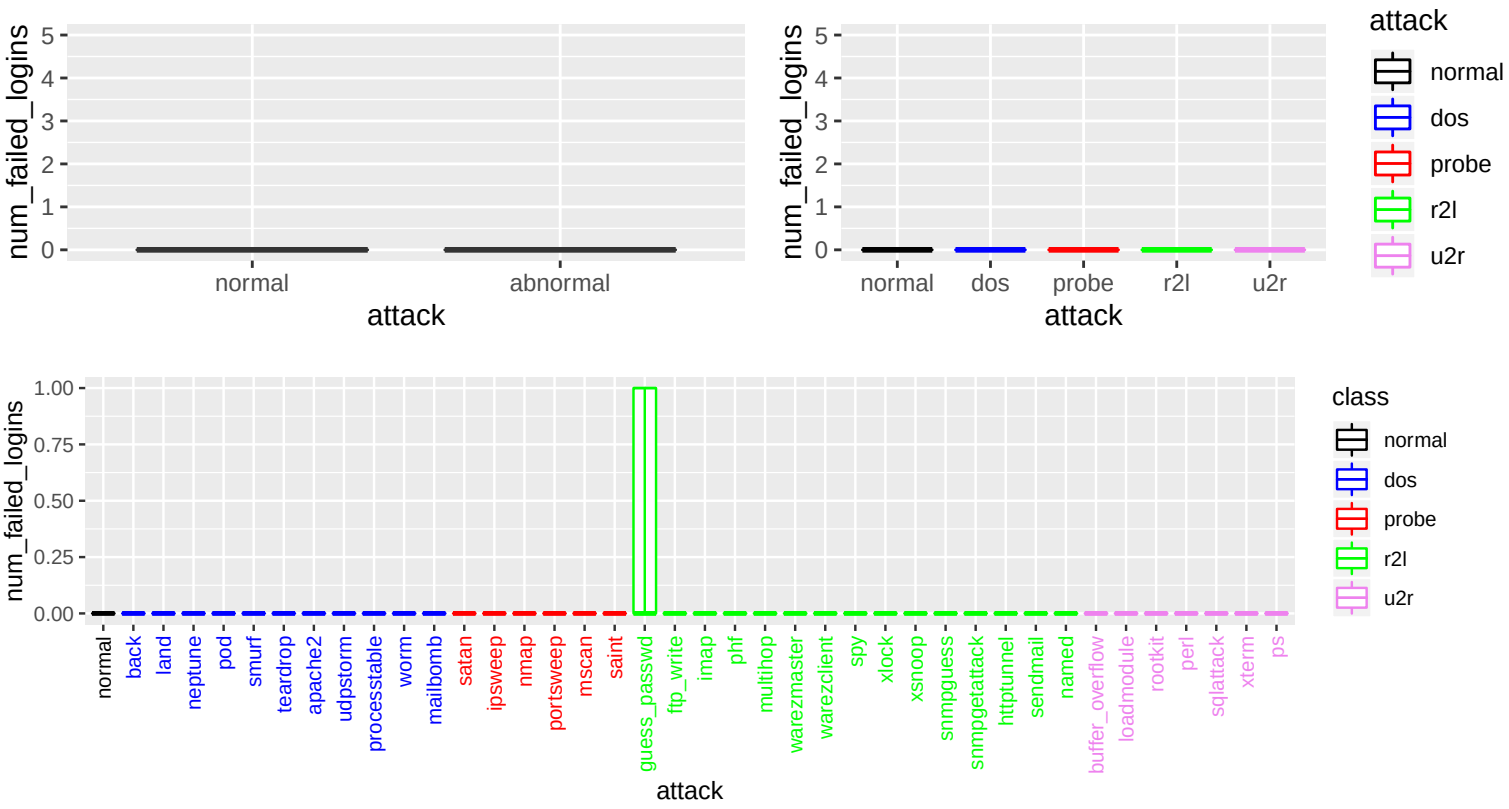


5. Urgent: The only attacks that seem to be related to values greater than 0 are ftp write, sqlattack and xterm. In all these three attacks, the urgent bit is used in the TCP segment.

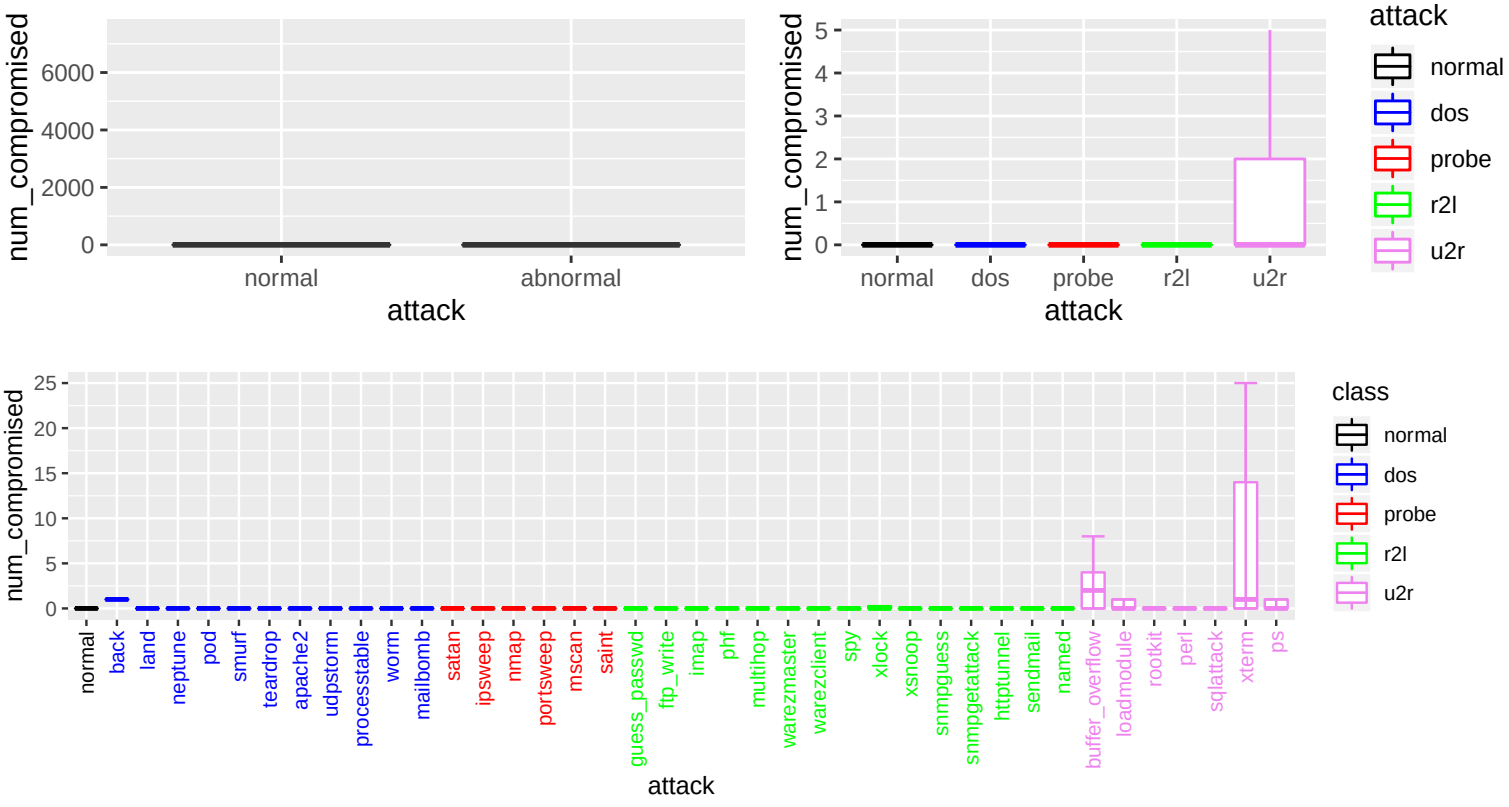


6. Hot: The number of hot indicators, seem to high is u2R attacks mainly. On breading down to individual attack types, we

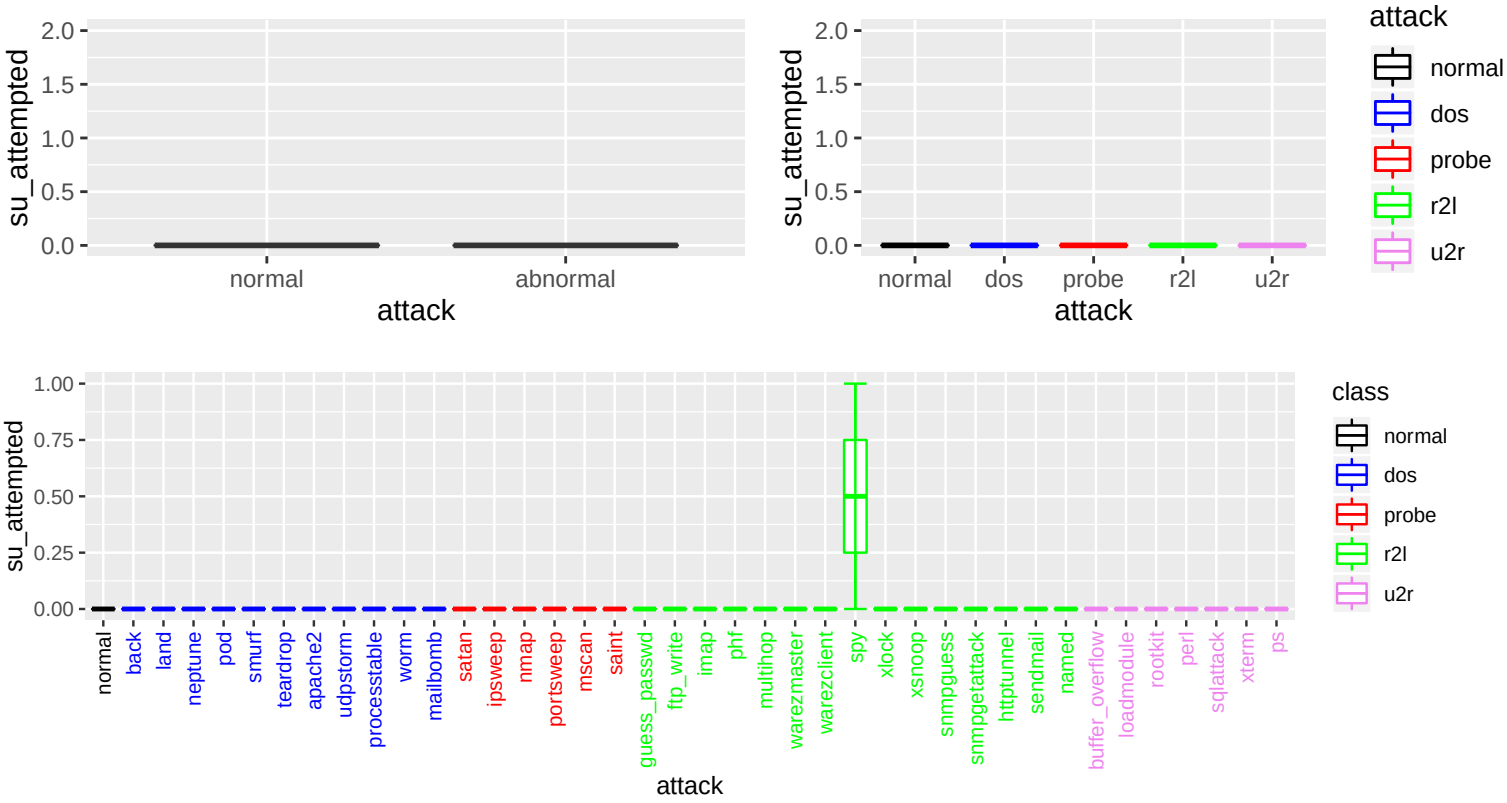
see that some of the R2L attacks have high hot indicators as well. Although the Warezclient attack under the R2L class has a median of 0, it has a high 3rd quartile value, exceeding 25.



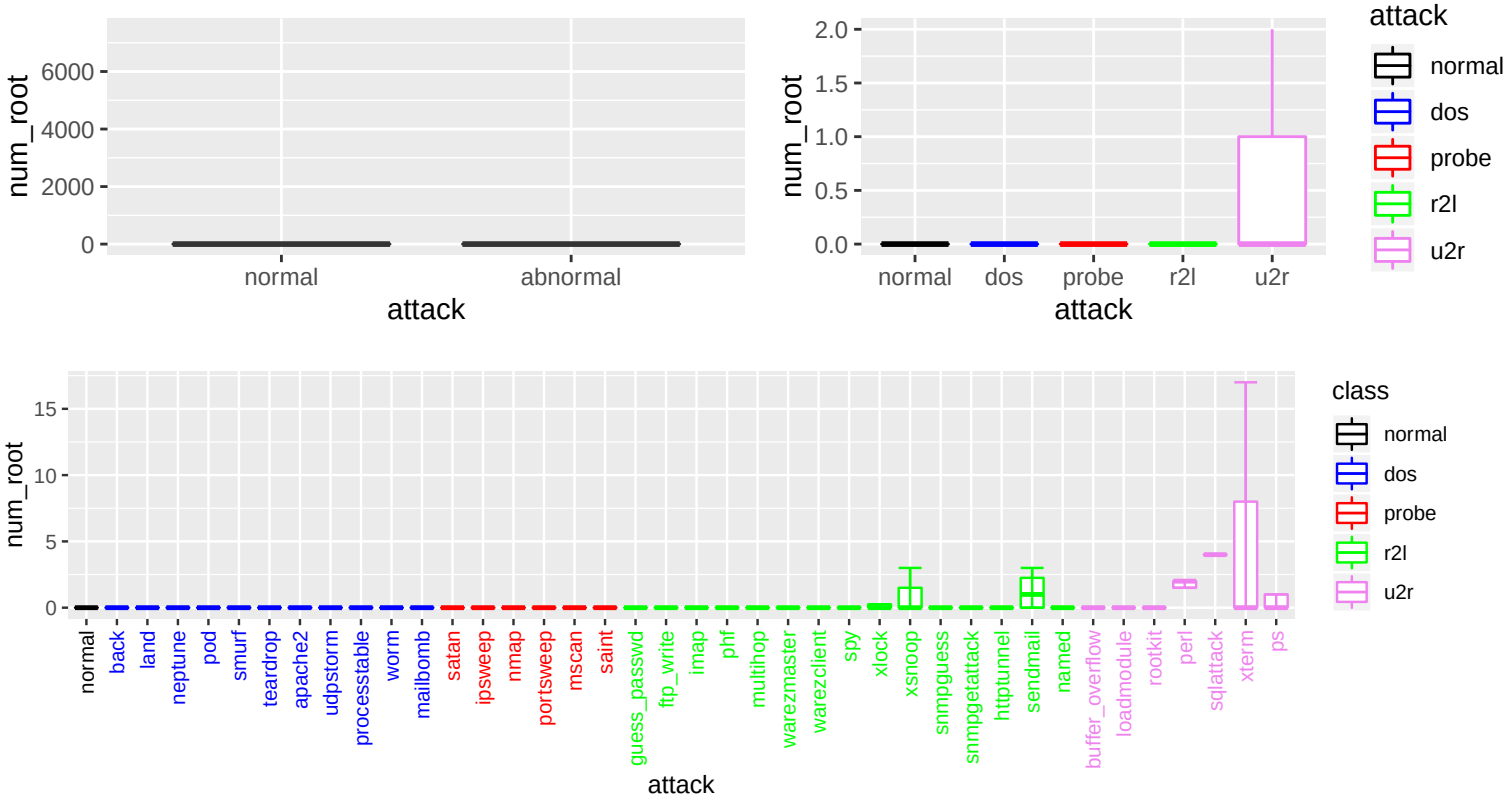
7. Number of Failed Logins: Guess Password attack is the only attack that has relatively high values of 'num_failed_logins', although its median is 0 just like all the other attacks.



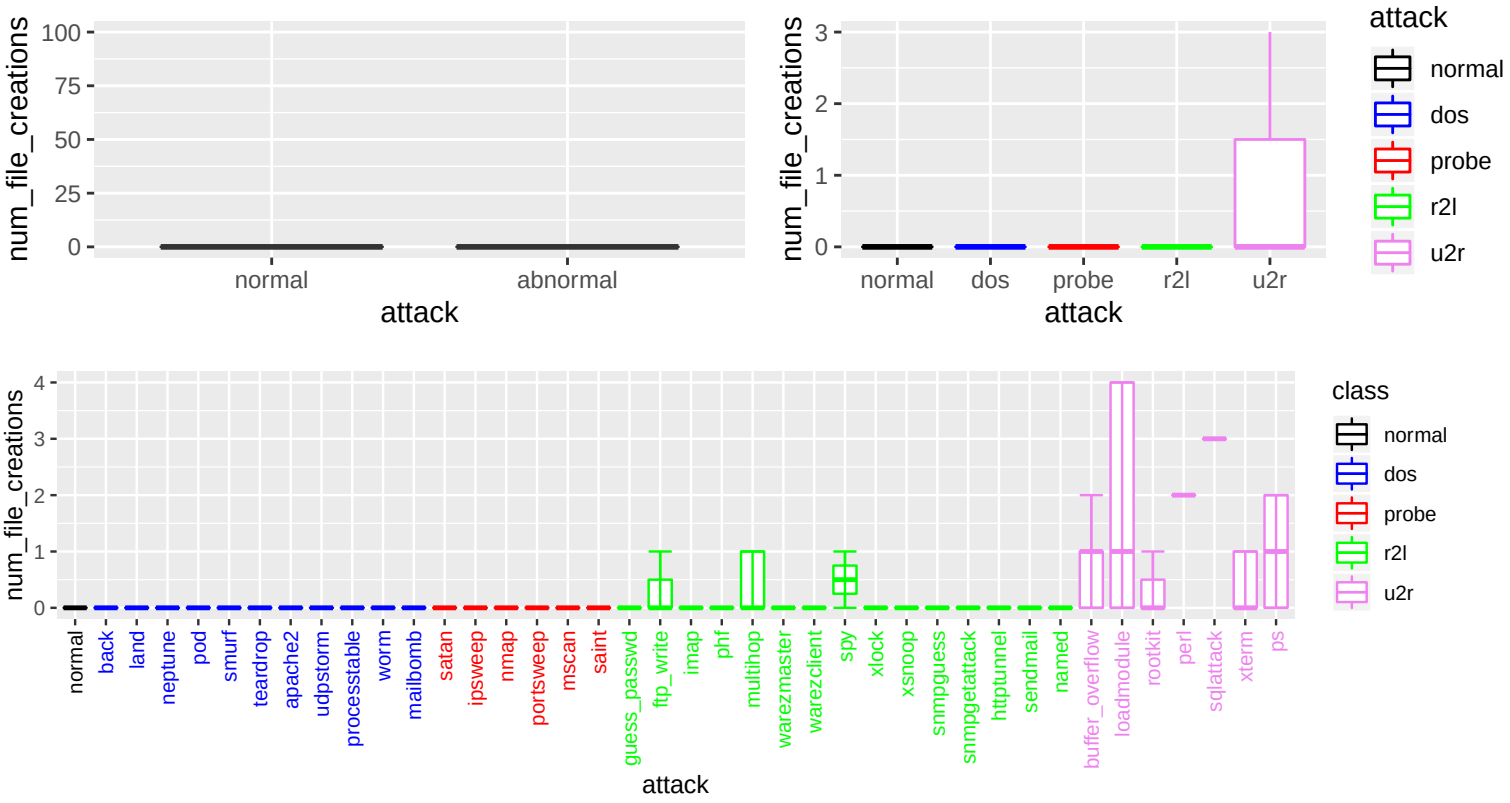
8. Number Compromised: The number of compromised conditions seems to be higher in U2R attacks overall, which can be seen when we break down the attack classes to individual types. Back attack under the DoS class almost always has 1 - 2 number of compromised conditions



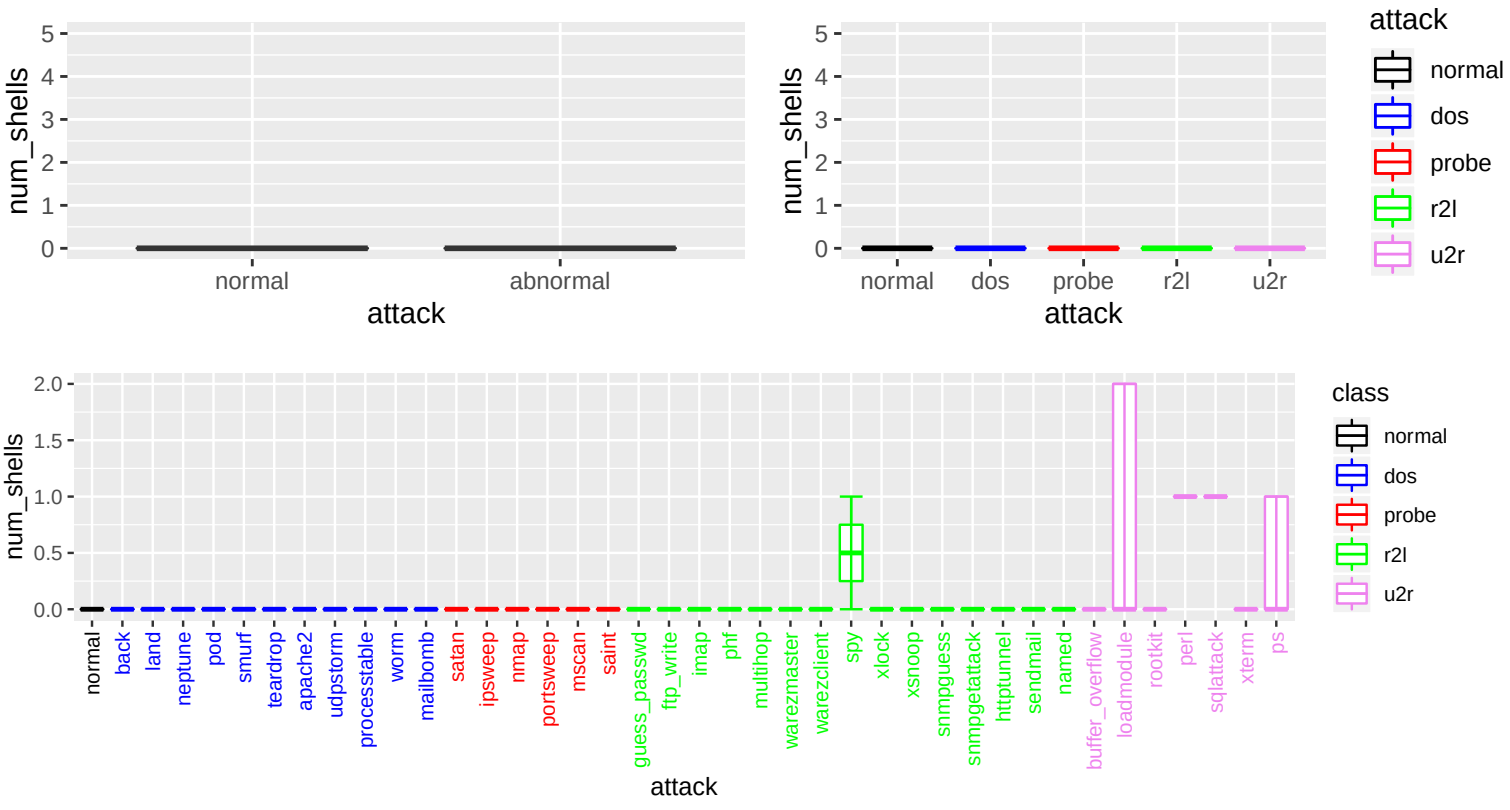
9. Super User Command attempt: Spy attack under the R2L attack class seems to be the only attack where the Super User command is attempted with a median rate of 0.5



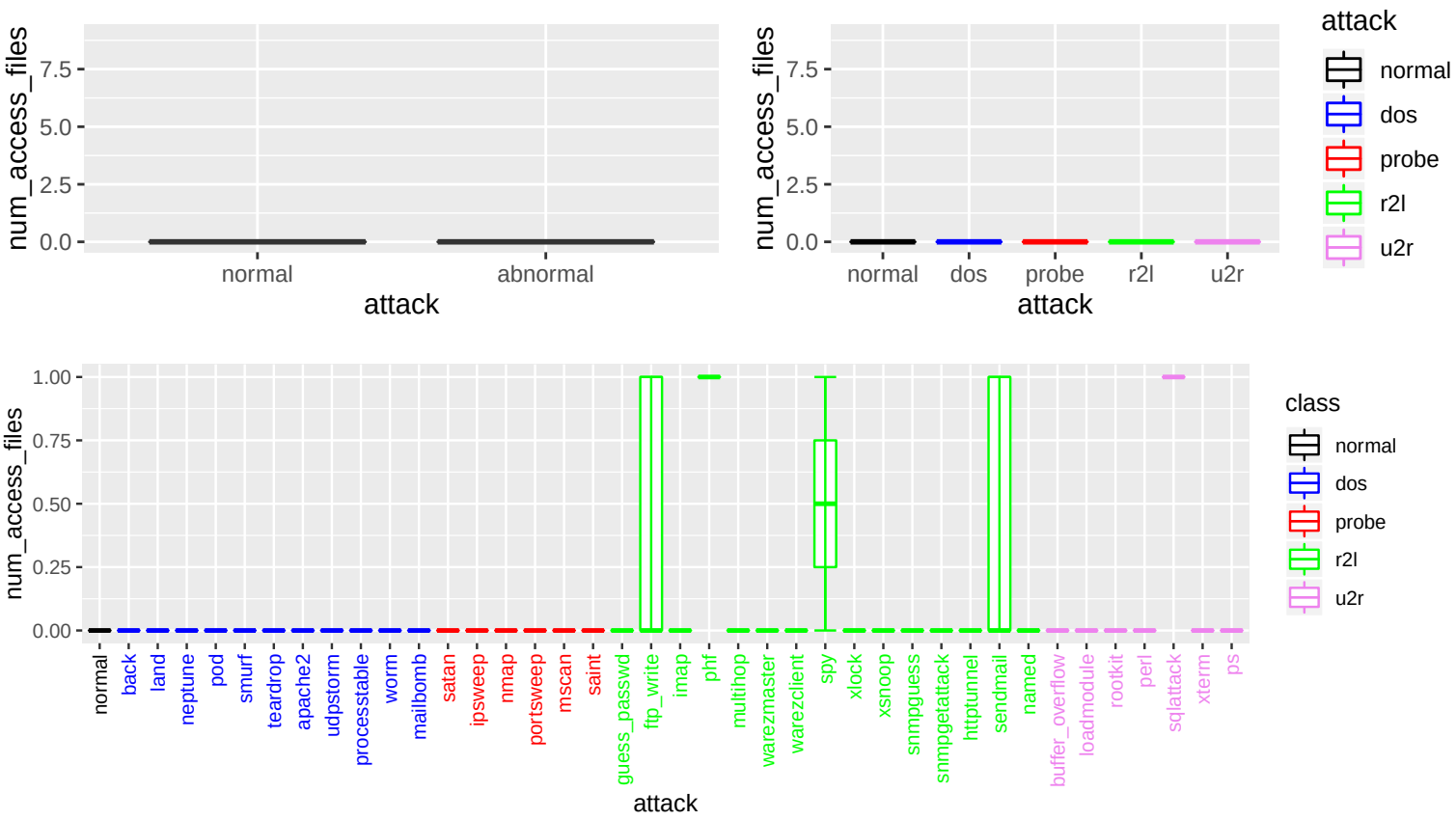
10. Number of Root accesses: Compared to other attack classes, U2R attack classes seem to have generally higher number of root accesses. Under the U2R class – Perl, SQL attack, Xterm and PS attacks have higher number of operations performed as a root. Xsnoop and Sendmail attacks under the R2L attacks also seem to have relatively high number of root accesses



11. Number of File Creations: U2R attack class generally has higher number of file creations compared to other attack classes. Almost all the attack types under this class have generally higher number of file creations associated to them, although, Rootkit and Xterm attacks have a median of 0. SQL attack and Perl attack have the highest medians at 3 and 2 respectively. FTP Write, multihop and Spy attacks under the R2L class are also somewhat associated to file creations.

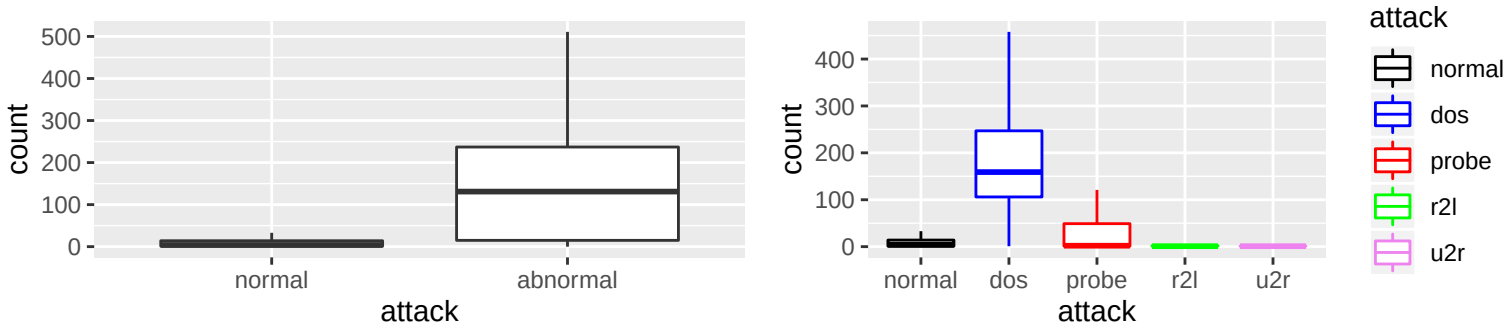


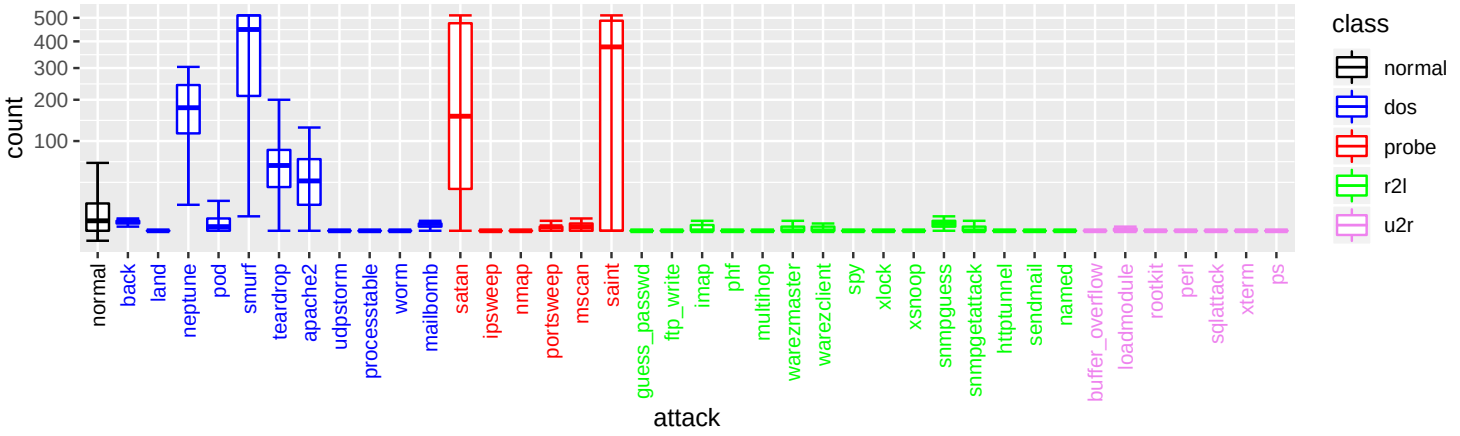
12. Number of Shell prompts: Perl and SQL attacks under the U2R class have the highest medians at 1 shell prompt. Although, Loadmodule and PS attacks have a median of 0, they have relatively higher 3rd quartile value of 2 and 1 respectively. Spy attack under R2L has the 2nd and 3rd quartiles at 0.25 and 0.75 respectively.



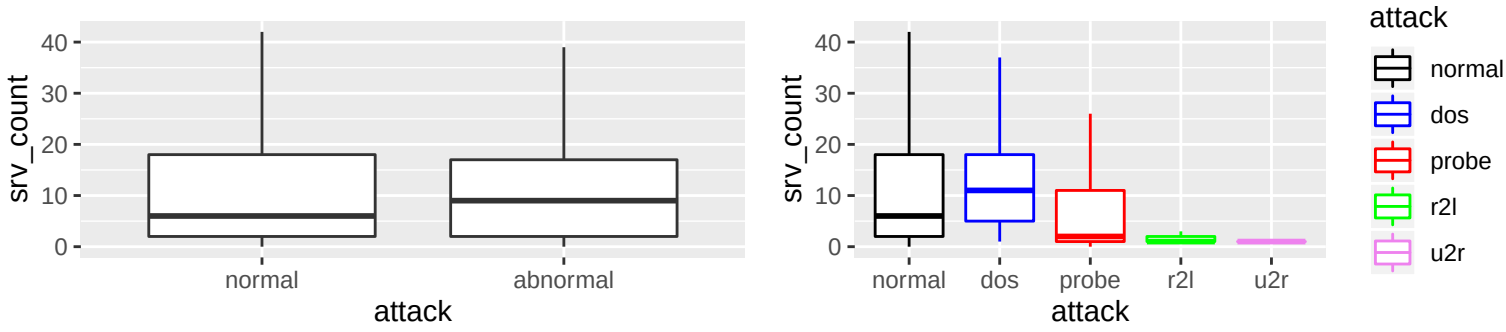
13. Number of operations on access control files: SQL attack under the U2R class and PHF attack under the R2L attack class almost always have 1 Operation on the access control files. FTP Write and Sendmail attacks have their median at 0. However, they do have operations on access control files some of the time.

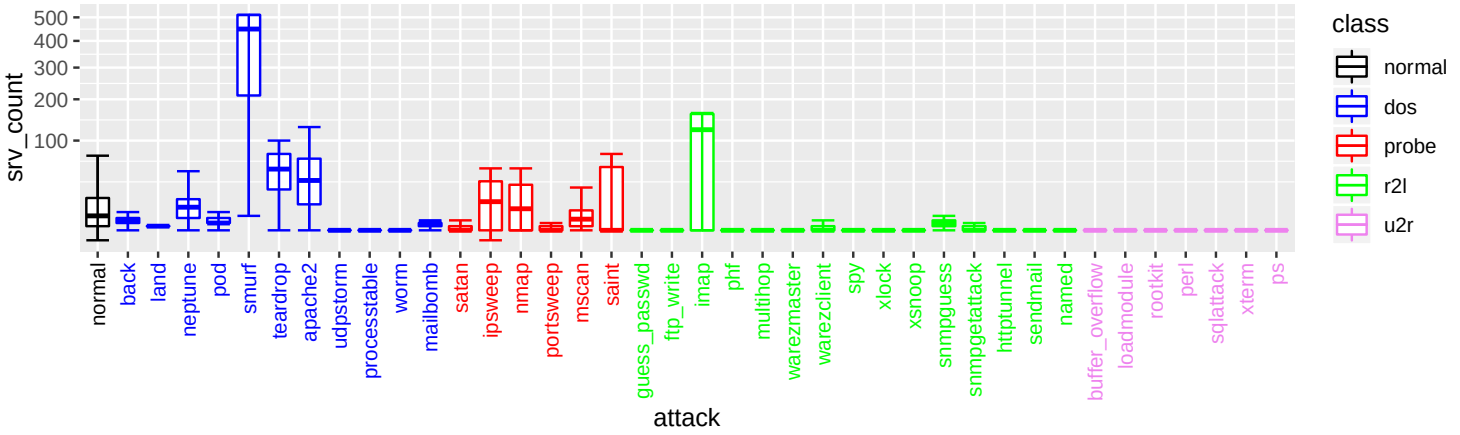
TIME RELATED TRAFFIC FEATURES:



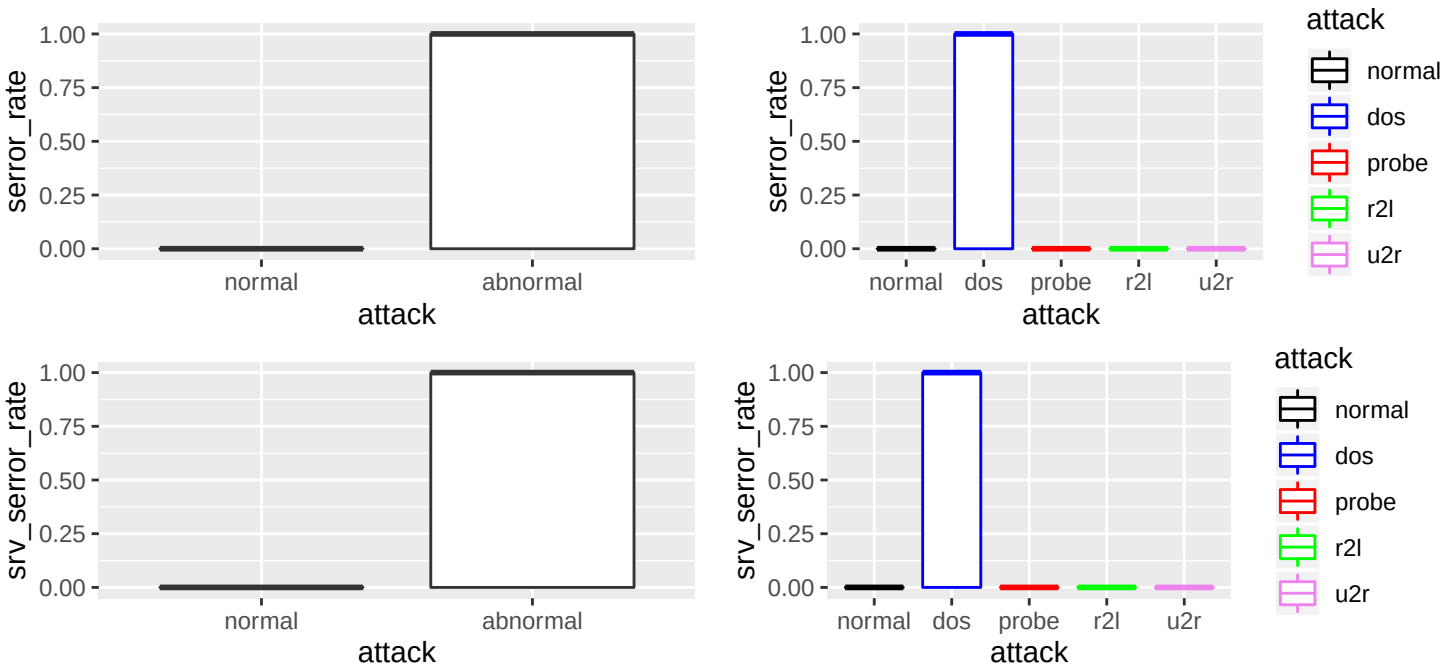


14. Count: Abnormal networks have, in general, higher values of count i.e higher number of connections to the same destination host as the current connection in the past two seconds. DoS attack class, in general, has higher values of count compared to other classes, which makes sense, because this attack class generally involves flooding the targeted machine or resource with superfluous requests. Apart form the DoS class, we see that SATAN and SAINT attacks under the Probe class also have high values of count. In both, SATAN and SAINT, every live system on a network is scanned, launching a set of probes designed to detect anything that could allow an attacker to gain unauthorized access

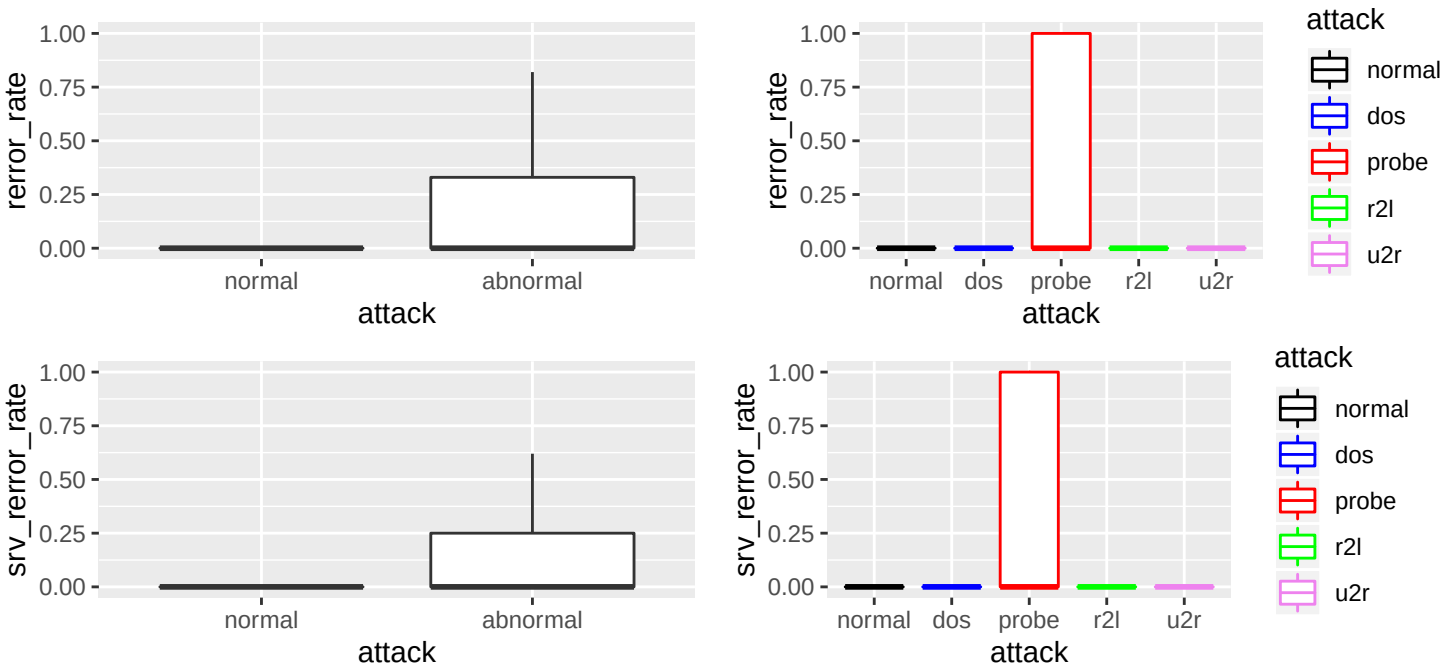




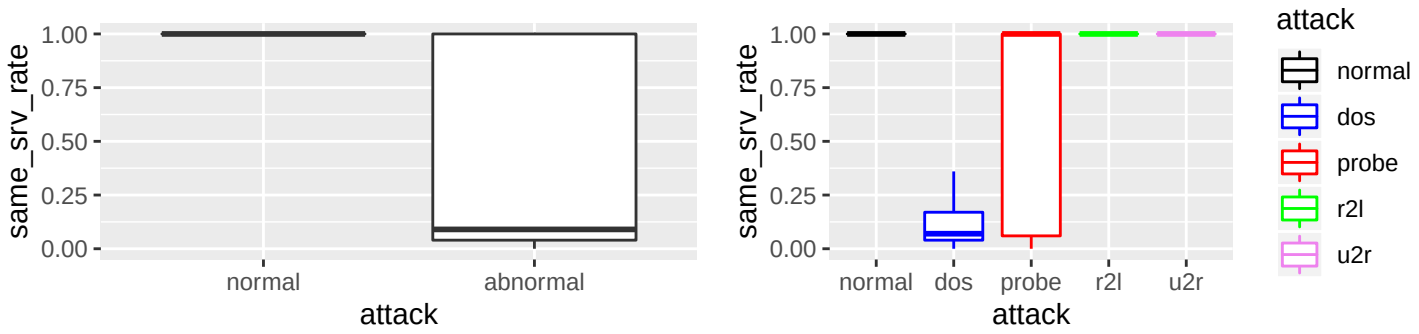
15. Service Count: Both U2R and R2L attacks, in general have lower values of Srv_count i.e lower number of connections to the same service (port number) as the current connection in the past two seconds as compared to other attacks and normal networks. However, the exceptions to this from the R2L class are Imap, snmp guess, snmp get attack and Warezclient attacks. Smurf attack has the highest median at around 450. Smurf is DoS type attack, in which large numbers of ICMP packets with the intended victim's spoofed source IP are broadcast to a network using an IP broadcast address. Teardrop, Apache2 and Neptune attacks under the DoS class and IP-Sweep, NMAP and MSCAN under the Probe class generally have high values of srv_count.



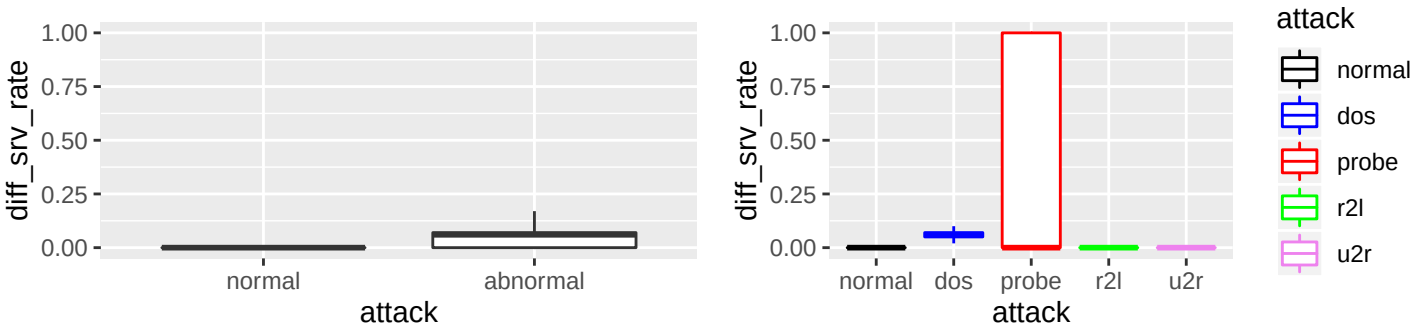
16. Serror_rate and 17. Srv_serror_rate i.e the percentage of connections that have SYN errors among the connections aggregated in count and srv_count respectively seem to be correlated. The Median for abnormalities and DoS attack class are at 100%



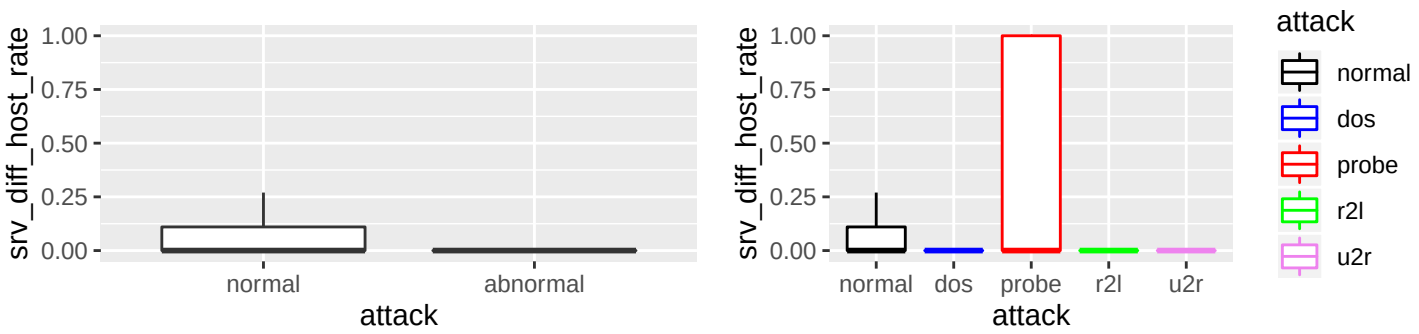
18.Rerror_rate and 19. Srv_rate i.e the percentage of connections that have 'REJ' errors among the connections aggregated in count and srv_count also seem to be somewhat correlated, with the third quartile exceeding 0.2 for abnormal connections and equal to 1 for Probe attack class



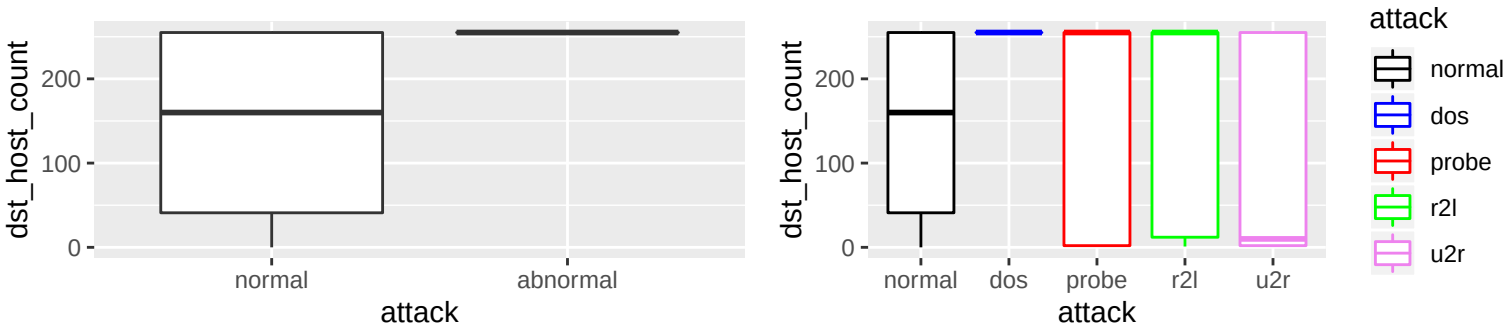
20. Same_srv_rate i.e the percentage of connections to the same service is almost always 100% for Normal networks, R2L attacks and U2R attacks. DoS attacks generally have lower values.

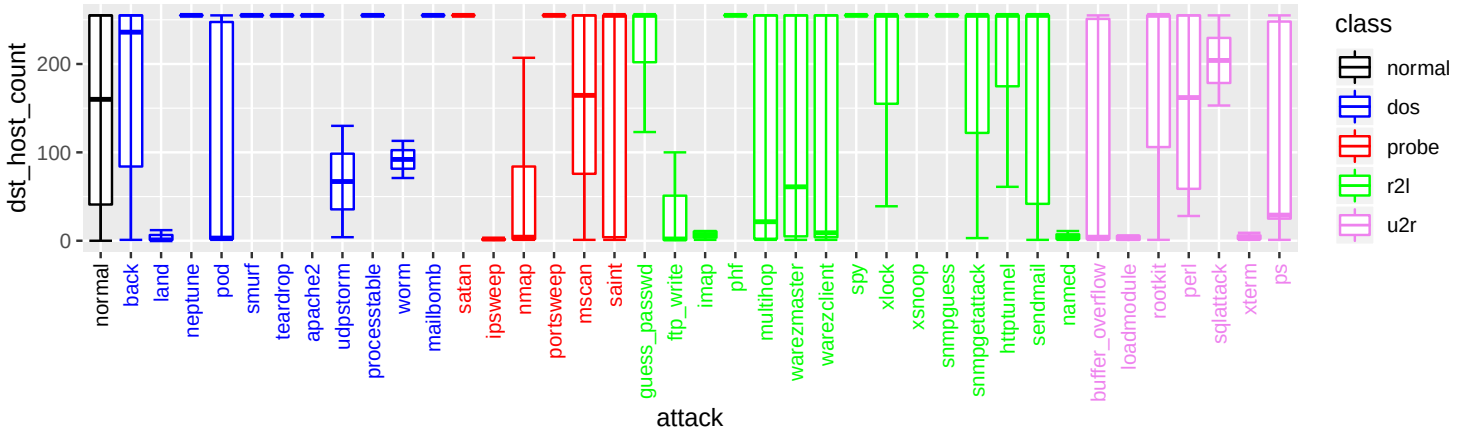


21. Diff_Srv_rate i.e the proportion of connections to different services is generally higher for abnormal connections with a median at over 5%. Among the attacks, Probe attacks have the highest IQR, with the 3rd Quartile at 100%



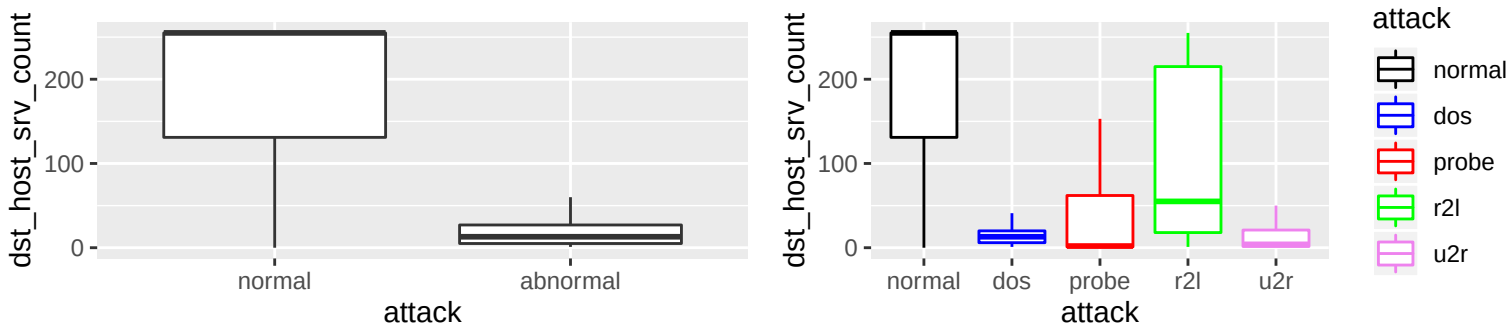
22. Srv_diff_host_rate i.e the proportion of connections to different hosts seems to be most of the times higher for Normal connections, although its median is at 0. Probe attack class has an IQR of 100%, while with rest of attacks are almost always 0

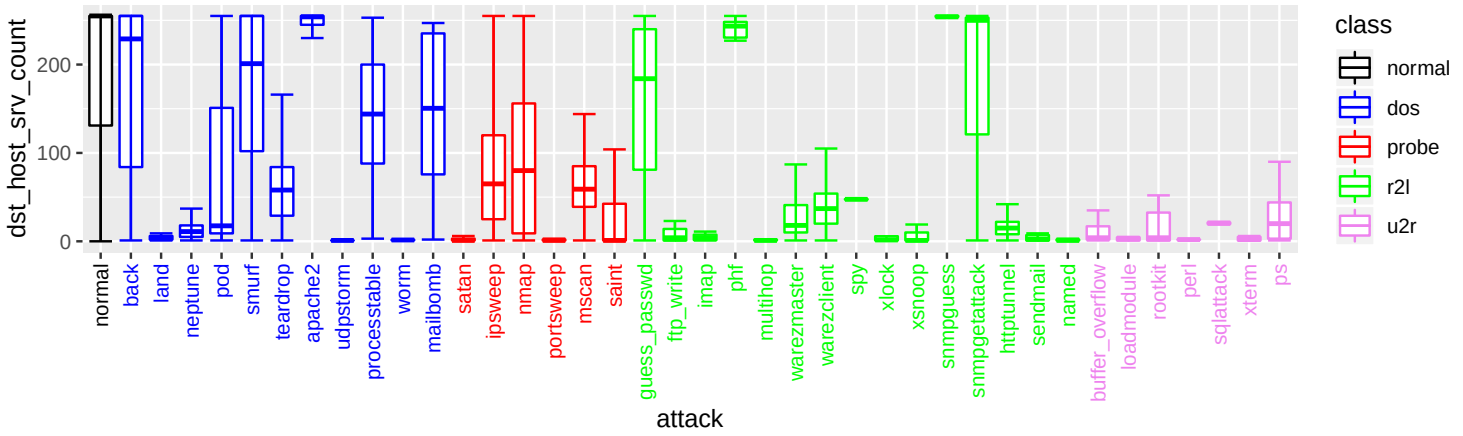




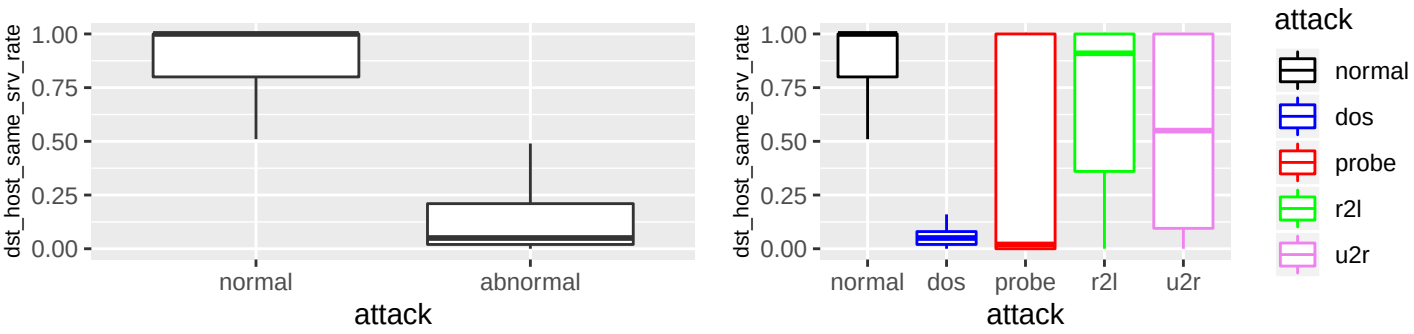
HOST BASED TRAFFIC FEATURES:

23. Dst_host_count: The number of connections having the same destination host IP address is high (almost always equal to 2^8 -1 = 255, which is the max number of IPs possible in a /8 network) for abnormal connections. It is almost always high for most of the DoS attacks expect land. Spy, Xsnoop, Snmpguess and Phf attacks under the R2L class and SATAN, Portsweep attacks under the Probe class also have Dst_host_count almost always equal to 255.

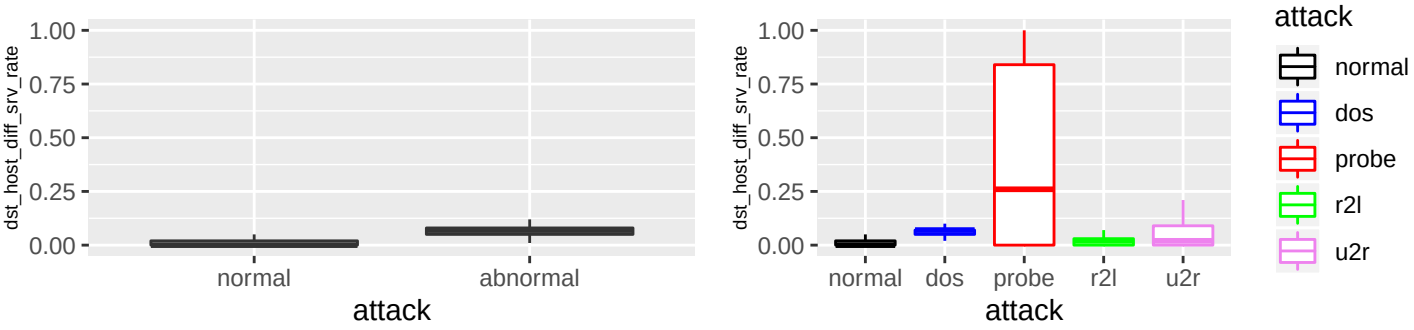




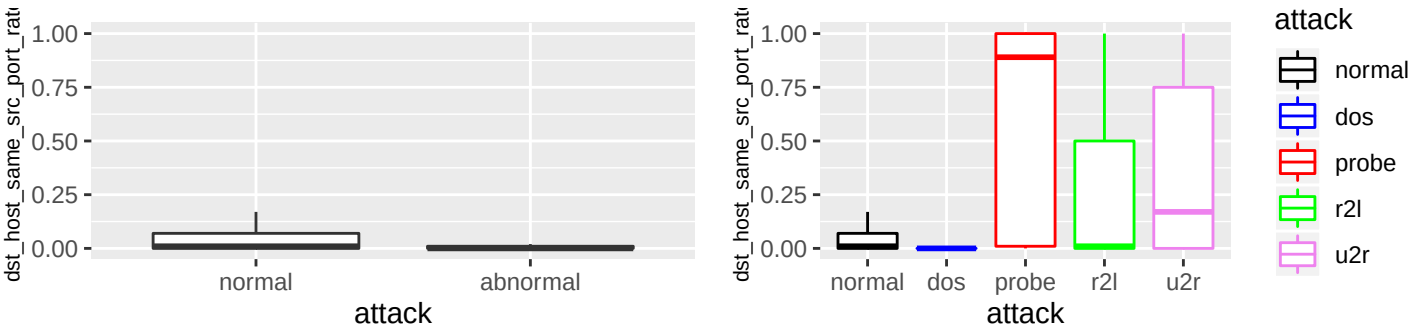
24. Dst_host_srv_count: The number of connections having the same port number is almost always high for normal network and considerably lower for abnormal networks. U2R and DoS attacks almost always have very low values. Again, because of the maximum number of IP possible in a /8 network is 255, the maximum value for Dst_host_srv_count is capped at 255



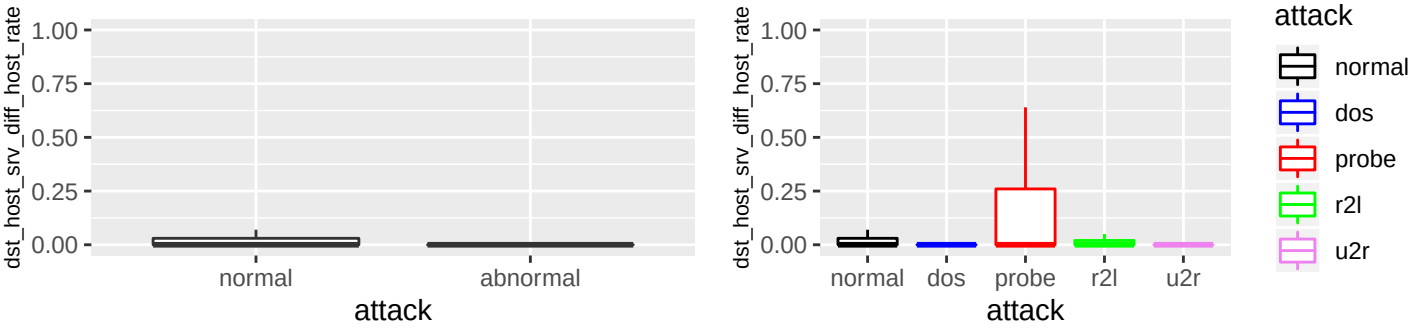
25. Dst_host_same_srv_rate (The percentage of connections to the same service, among the connections in dst_host_count) is high for most normal networks as well as to R2L and U2R attack classes. DoS attack class, however, remains closer to 0.



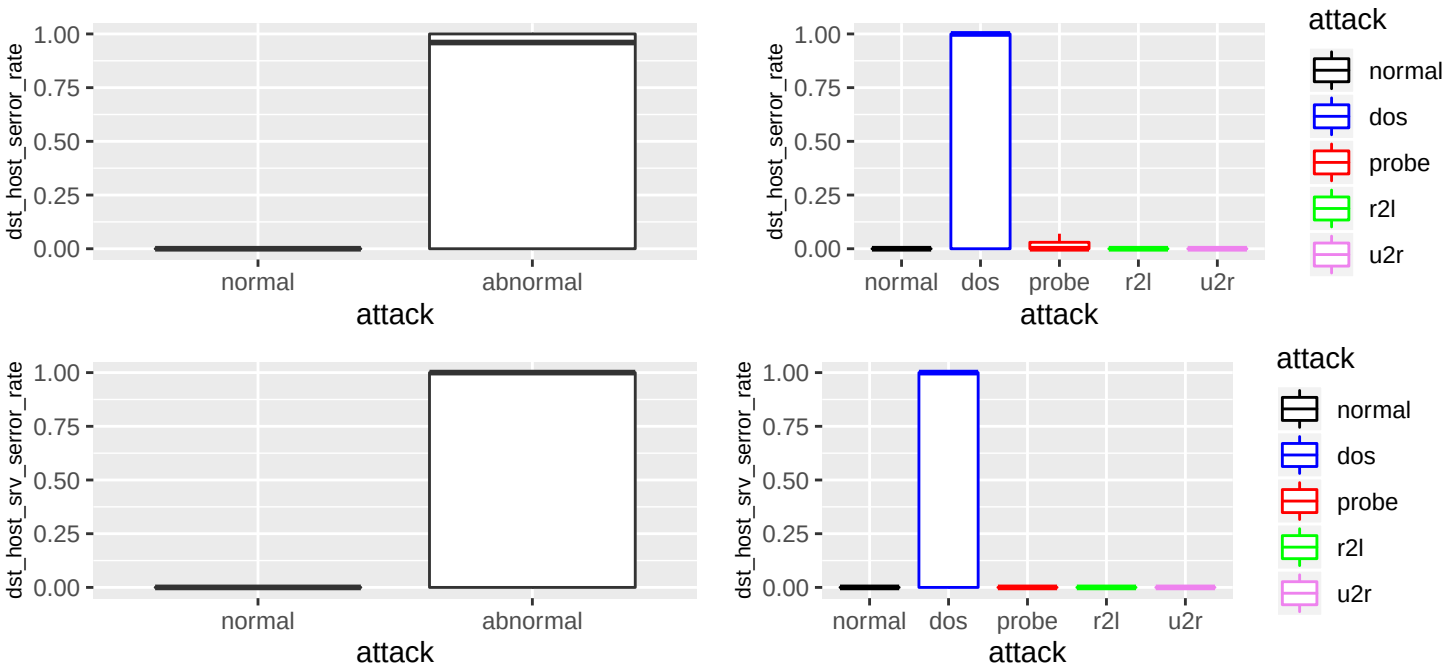
26. Dst_host_diff_srv_rate (The percentage of connections to different services, among the connections aggregated in dst_host_count) is low at around 0 for most normal networks, R2L , DoS and U2R attack classes. Probe attacks have a larger median with a larger range



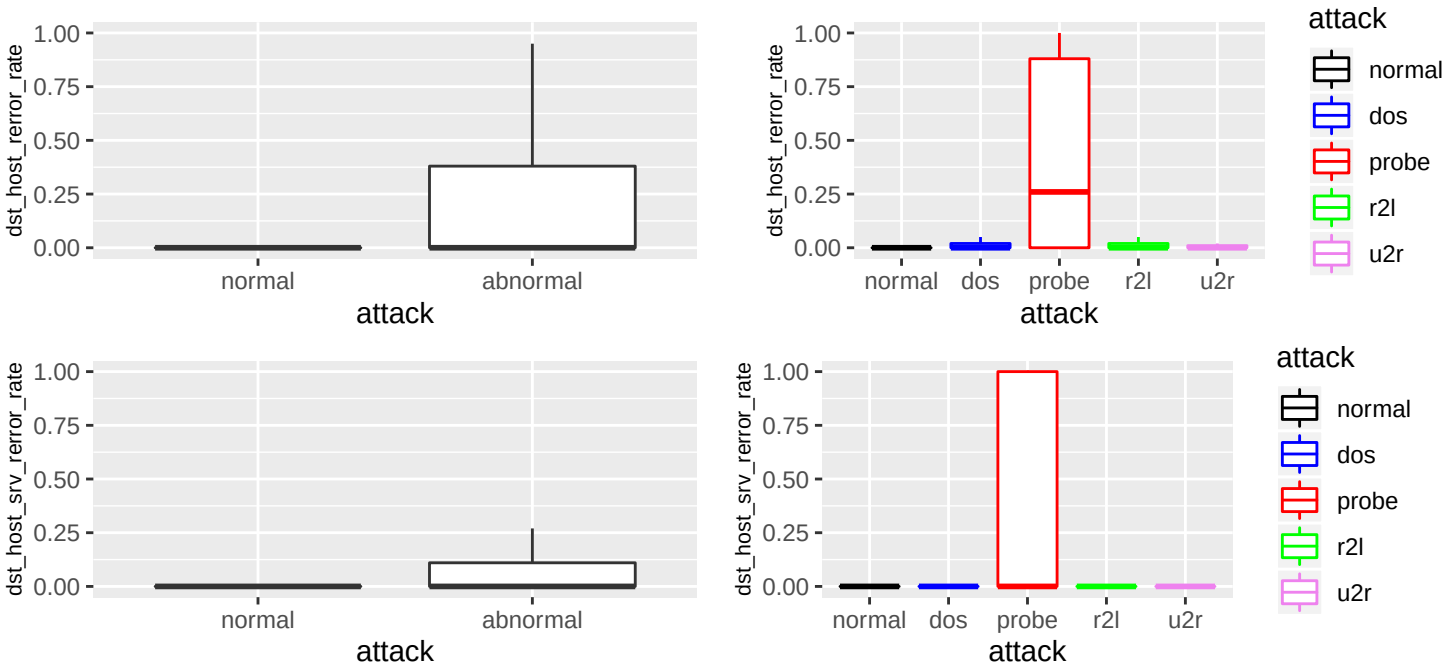
27. Dst_host_same_src_port_rate i.e the percentage of connections to the same source port, among the connections aggregated in dst_host_srv_count is almost always low for DoS attack class. It is highest in general for Probe attacks.



28. Dst_host_srv_diff_host_rate i.e the proportion of connections to different destination machines, among the connections aggregated in dst_host_srv_count, is almost always zero for attacks: DoS, R2L and U2R as well as Normal networks.

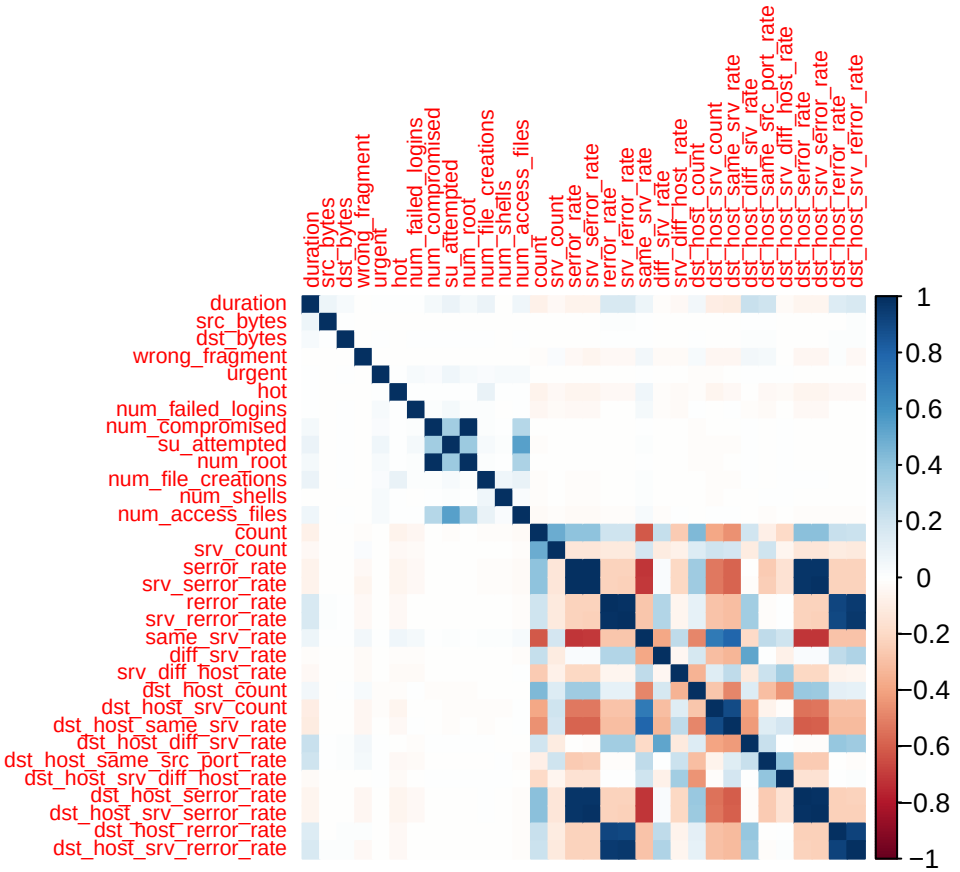


29. Dst_host_serror_rate (percentage of connections that have SYN errors among the connections aggregated in dst_host_count) and 30. Dst_host_srv_s error_rate (percentage of connections that have SYN errors among the connections aggregated in dst_host_srv_count) seem to have a high correlation. Both are almost always 0 for normal networks and all attack classes except for DoS



31. Dst_host_rerror_rate (percentage of connections that have REJ errors among the connections aggregated in dst_host_count) and 32. Dst_host_srv_rerror_rate (The percentage of connections that have REJ errors among the connections aggregated in dst_host_srv_count) also seem to be heavily correlated, with both attributes being almost always 0 for normal networks and all attack classes except for Probe attack class

CORRELATION PLOT OF NUMERICAL FEATURES:



Looking at the correlation plot for the numerical attributes, we see, as suspected from the boxpots, high postive correlation between pairs:

1. serror_rate and srv_serror_rate
2. rerror_rate and srv_rerror_rate
3. dst_host_serror_rate and dst_host_srv_serror_rate
4. dst_host_rerror_rate and dst_host_srv_rerror_rate

In fact, we find that out of the eight attributes listed above; i. serror_rate, srv_serror_rate (1), dst_host_serror_rate and dst_host_srv_serror_rate (3) are all heavily correlated with each other. ii. In the same manner, rerror_rate, srv_rerror_rate (2), dst_host_rerror_rate and dst_host_srv_rerror_rate (4) are all heavily correlated with each other.

We also find high positive correlation between dst_host_same_srv_count and dst_host_same_Srv_rate

Strong negative correlation can be found between:

1. same_srv_rate and serror_rate
2. same_srv_rate and srv_serror_rate
3. same_srv_rate and dst_host_srv_serror_rate
4. same_srv_rate and dst_host_rerror_rate

6 Data Modelling

The edx datasets for anomaly detection and classification are further split into training and test datasets. The train set will be the subset used to train different models. The test set will be the subset used to test and evaluate the trained models.

```
#Creating Train and Test sets
set.seed(1, sample.kind="Rounding")
train_index <- createDataPartition(y = edx_set$attack, times = 1, p = 0.8, list = FALSE)
train_set <- edx_set[train_index,]
test_set <- edx_set[-train_index,]

#Filtering out the numerical features
num_train <- as.data.frame(train_set[,sapply(train_set, is.numeric)])
num_test <- as.data.frame(test_set[,sapply(test_set, is.numeric)])

#Filtering out the categorical features
cat_train <- as.data.frame(train_set[,!sapply(train_set, is.numeric)])%>%
  select(-c(attack, detect, attack_class))
cat_test <- as.data.frame(test_set[,!sapply(test_set, is.numeric)])%>%
  select(-c(attack, detect, attack_class))
```

Once the model has been finalized, the edx sets will be used to train the final models and make the final predictions on the validation set.

We would be performing principal component analysis to try and reduce the number of dimensions and thereby reducing the computation time. It would also helpful to remove the redundant features.

Before we perform principal component analysis, it is essential to standardize the numerical features, so that the variables with the biggest scales don't overwhelm the PCA.

```
# Standardizing numerical data on anomaly and classify datasets
mean_train <- apply(num_train, 2, function(x){mean(x)})
sd_train <- apply(num_train, 2, function(x){sd(x)})
std_features <- apply(num_train,2,function(x){(x-mean(x))/(sd(x))})
```

The 'PCAmix' function under the 'PCAmixdata' package can ne used to perform principal component analysis on a dataset with a mix of qualititaive and quantitative features.

```
# Performing PCA on the the mixed datatypes
pca_mix<-PCAmixdata::PCAmix(X.quanti=num_edx,X.quali=cat_edx,rename.level=TRUE,graph=FALSE)
```

	Eigenvalue	Proportion	Cumulative
dim 1	8.5	7.2	7.2
dim 2	5.9	5.0	12.2
dim 3	3.3	2.8	15.1
dim 4	2.8	2.4	17.4
dim 5	2.7	2.3	19.7
dim 6	2.5	2.1	21.9

	Eigenvalue	Proportion	Cumulative
dim 112	0.02	0.02	100
dim 113	0.01	0.01	100
dim 114	0.01	0.01	100
dim 115	0.01	0.01	100
dim 116	0.00	0.00	100
dim 117	0.00	0.00	100

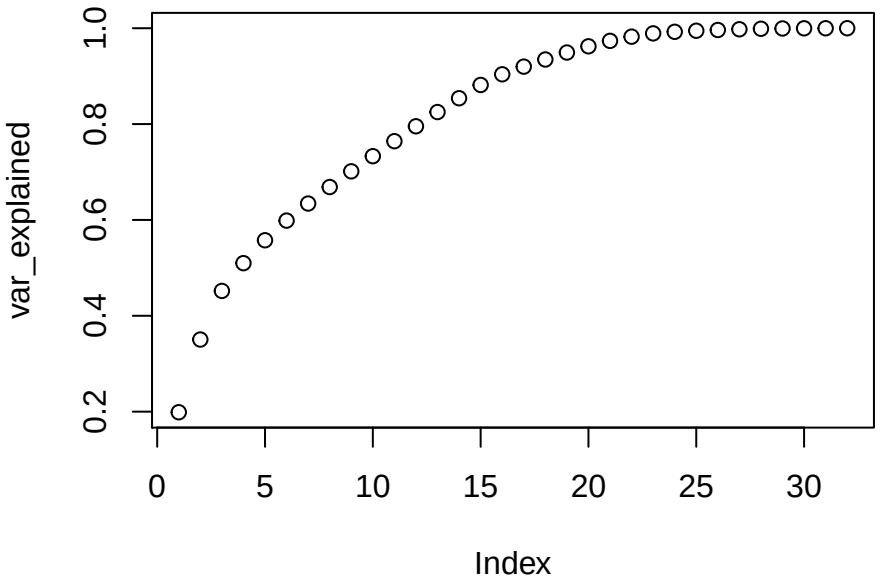
	Eigenvalue	Proportion	Cumulative
dim 80	0.99	0.84	88
dim 81	0.99	0.84	89
dim 82	0.98	0.83	90
dim 83	0.93	0.79	91
dim 84	0.89	0.75	92
dim 85	0.89	0.75	92

Because of the various levels/factors for each qualitative feature, the total number of Principal Components needed to explain all the variance has increased accordingly to 117. We see that 90% of the variance is explained by the first 82 principal components. Because of this vast number of dimensions needed (more than the number of dimensions in the dataset itself), Eigen-Value Based PCA on all the predictors cannot be used for dimension reduction for this dataset

We can perform Principal Component Analysis via Singular Value Decomposition of the numerical features to try and reduce the number of features.

```
# Performing SVD-PCA on the numerical parameters alone on the anomaly dataset
pc <- prcomp(std_features, center = F, scale. = F)
var_explained <- cumsum(pc$sdev^2/sum(pc$sdev^2))
var_explained
```

```
##  [1] 0.20 0.35 0.45 0.51 0.56 0.60 0.63 0.67 0.70 0.73 0.76 0.80 0.83 0.85 0.88
## [16] 0.90 0.92 0.93 0.95 0.96 0.97 0.98 0.99 0.99 0.99 1.00 1.00 1.00 1.00 1.00
## [31] 1.00 1.00
```



We see that around 90% of the variance is explained by the first sixteen principal components. Thus, we can effectively halve the number of numerical features from 32 to 16 principal components.

Just like we've obtained PCA components on training set, we'll get another bunch of components on testing set. We should not combine the train and test set to obtain PCA components of whole data at once. If we combine the train and test set to obtain PCA components of whole data, this would violate the entire assumption of generalization since test data would get 'leaked' into the training set. In other words, the test data set would no longer remain 'unseen'. Also, we should not perform PCA on test and train data sets separately, because, the resultant vectors from train and test PCAs will have different directions (due to unequal variance). Due to this, we'll end up comparing data registered on different axes.

The resulting vectors from train and test data should have the same axes. Thereby, we should do exactly the same transformation to the test set as we did to the training set, including the center and scaling feature.

```
#Chosing the first 16 principal components
train.data <- data.frame(train_set[,!sapply(train_set, is.numeric)], pc$x[,1:16])
train.data_detect <- train.data %>% select(-c(attack, attack_class))
train.data_attack_class <- train.data %>% select(-c(attack, detect))

#Selecting only numerical features from the test set
```

```
num_test <- as.data.frame(test_set[,sapply(test_set, is.numeric)])

#Centering and Scaling the test set using the mean and Sd of Train set
std_features_test <- mapply(function(x,y,z){(x-y)/z},num_test,mean_train,sd_train)

# Predicting the principal components of the numerical features in test set
test.data <- predict(pc, std_features_test)
test.data <- data.frame(test_set[,!sapply(test_set, is.numeric)],test.data[,1:16])
test.data <- test.data %>% select(-c(attack, attack_class, detect))
```

We still have a considerable number of predictors (twenty-four) even after dimension reduction. Since Random Forest and Random Trees are most suited to develop models with high dimensions, both will be used.

6.1 Random Forest

The Ranger function is a fast implementation of random forests, particularly suited to high dimensional data.

6.1.1 Network Anomaly Detection

Using Random Forest for Network Anomaly Detection:

```
options(digits=4, scipen=999)
#Training
train_rf_detect <- ranger(detect ~ .,
                          data = train.data_detect)

#Making predictions
pred_rf_detect <- predict(train_rf_detect, data = test.data)$predictions

#Getting Accuracy
a1 <- confusionMatrix(pred_rf_detect,
                      test_set$detect)$overall["Accuracy"]

#Getting Sensitivity
se1<- confusionMatrix(pred_rf_detect,
                      test_set$detect)$byClass["Sensitivity"]

#Getting Specificity
sp1 <- confusionMatrix(pred_rf_detect,
                       test_set$detect)$byClass["Specificity"]
```

```
results_rf_detect <- data_frame(Goal = "Anomaly Detection",
                                Method = "Random Forest",
                                Accuracy = a1, Sensitivity = se1, Specificity = sp1)

#Displaying the results
data.frame(results_rf_detect) %>% knitr::kable()
```

Goal	Method	Accuracy	Sensitivity	Specificity
Anomaly Detection	Random Forest	0.9926	0.9939	0.9911

We get an Accuracy of 99.26%, Sensitivity of 99.39% and Specificity of 99.11%

6.1.2 Network Attack Classification

Using Random Forest for Network Anomaly Classification:

```
#Training
train_rf_attack_classify <- ranger(attack_class ~ .,
                                   data = train.data_attack_class)

#Making predictions and getting the Accuracy
a3 <- confusionMatrix(predict(train_rf_attack_classify, data = test.data)$predictions,
                      test_set$attack_class)$overall["Accuracy"]
```

```
results_rf_classify <- data_frame(Goal = "Attack Classification",
                                  Method = "Random Forest",
                                  Accuracy = a3)

#Displaying the results
data.frame(results_rf_classify) %>% knitr::kable()
```

Goal	Method	Accuracy
Attack Classification	Random Forest	0.9923

We get an Accuracy of 99.23%

6.2 Decision Tree

Rpart package's implementation of the Decision Tree will be used. The default value of the complexity parameter is 0.01. However, a Complexity parameter of 0 will be used for both: Anomaly Detection and Attack Classification to get better performance.

6.2.1 Network Anomaly Detection

Using Random Tree for Network Anomaly Detection:

```
#Training
train_rpart_detect <- train(detect ~ .,
                         method = "rpart",
                         tuneGrid = data.frame(cp = 0),
                         data = train.data_detect)

#Making predictions
pred_rpart_detect <- predict(train_rpart_detect, test.data)

#Getting Accuracy
a2 <- confusionMatrix(pred_rpart_detect,
                      test_set$detect)$overall["Accuracy"]

#Getting Sensitivity
se2 <- confusionMatrix(pred_rpart_detect,
                       test_set$detect)$byClass[("Sensitivity")]

#Getting Specificity
sp2 <- confusionMatrix(pred_rpart_detect,
                       test_set$detect)$byClass[("Specificity")]

results_rpart_detect <- data_frame(Goal = "Anomaly Detection",
```

```
                                    Method = "Decision Tree",
                                    Accuracy = a2,
                                    Sensitivity = se2,
                                    Specificity = sp2)

#Displaying the results
data.frame(results_rpart_detect) %>% knitr::kable()
```

Goal	Method	Accuracy	Sensitivity	Specificity
Anomaly Detection	Decision Tree	0.9887	0.9898	0.9874

We get an Accuracy of 98.87%, Sensitivity of 98.98% and Specificity of 98.74%

6.2.2 Network Attack Classification

Using Random Tree for Network Anomaly Classification:

```
#Training
train_rpart_attack_classify <- train(attack_class ~ .,
                             method = "rpart",
                             tuneGrid = data.frame(cp = 0),
                             data = train.data_attack_class)

#Making predictions and getting the Accuracy
a4 <- confusionMatrix(predict(train_rpart_attack_classify, test.data),
                      test_set$attack_class)$overall["Accuracy"]

results_rpart_classify <- data_frame(Goal = "Attack Classification",
                                     Method = "Decision Tree",
                                     Accuracy = a4)

#Displaying the results
data.frame(results_rpart_classify) %>% knitr::kable()
```

Goal	Method	Accuracy
Attack Classification	Decision Tree	0.9878

We get an Accuracy of 98.78%

7 Results

It is observed that Random Forest gives us higher Accuracy, Specificity and Sensitivity values for Anomaly Detection. Radom Forest also gives us higher accuracy values for Network Attack Classification. Thereby, Ranger function's implementation of random forests is chosen as the final model to be trained on the edx set.

TO train the edx set, firstly, the numerical features of the edx set will be scaled and centred. Following this, PCA will be performed on these scaled numerical features and only the first 16 Principal Components will be retained.

Column no: 20 i.e 'num_outbound_cmds' will be removed from the validation set, as was done for the edx set during the Exploratory Data Analysis stage.

Similar to what was done to the Test set while developing the model, we will be scaling and centering the validation set using the mean and standard deviation of the Edx set. We will then perform SVD-PCA on the numerical features of the Edx set and predict the principal components of the numerical features of the validation set using the predict function. Only the first 16 predicted principal components of the validation set will be retained.

```r
#FIltering out the numerical features of the edx set
num_edx <- as.data.frame(edx_set[,sapply(edx_set, is.numeric)])

#Standardizing the numerical features of edx_set
mean_edx <- apply(num_edx, 2, function(x){mean(x)})
sd_edx <- apply(num_edx, 2, function(x){sd(x)})
std_features_edx <- apply(num_edx,2,function(x){(x-mean(x))/(sd(x))})

# Perfroming PCA on the standardized numerical features of Edx set
pc <- prcomp(std_features_edx, scale. = F, center = F)

#Chosing only the first 16 principal components
edx.data <- data.frame(edx_set[,!sapply(edx_set, is.numeric)], pc$x[,1:16])

#Final edx set for anomaly detection
edx.data_detect <- edx.data %>% select(-c(attack, attack_class))

#FInal edx set for Attack classification
edx.data_attack_class <- edx.data %>% select(-c(attack, detect))

#Removing column 20 i.e 'num_outbound_cmds'
validation_set <- validation_set[,-20]

#Selecting only numerical features
num_validation <- as.data.frame(validation_set[,sapply(validation_set, is.numeric)])

#Centering and Scaling the test set using the mean and Sd of edx sets
std_features_validation <- mapply(function(x,y,z){(x-y)/z},num_validation,mean_edx,sd_edx)

# Predict PCs of Validation set
validation.data <- predict(pc, std_features_validation)

# Selecting the first 16 predicted PCs
validation.data <- data.frame(validation_set[,!sapply(validation_set, is.numeric)],validation.data[,1:16])
validation.data <- validation.data %>% select(-c(attack, attack_class, detect))
```

We will then use the chosen model: Random Forest Ranger function to train on the Edx set.

7.1 Network anomaly Detection

Using the edx set to train the final model for anomaly Detection and then making the predictions on the validation set:

```r
edx_rf_detect <- ranger(detect ~ ., data = edx.data_detect)

#Making predictions
pred_detect <- predict(edx_rf_detect, data = validation.data)$predictions

#Getting Accuracy
acc_detect <- confusionMatrix(pred_detect,
                      validation_set$detect)$overall["Accuracy"]

#Getting Sensitivity
sen_detect <- confusionMatrix(pred_detect,
                      validation_set$detect)$byClass["Sensitivity"]

#Getting Specificity
spec_detect <- confusionMatrix(pred_detect,
                      validation_set$detect)$byClass["Specificity"]
```

```
results_detect <- data_frame(Goal = "anomaly Detection",
                                    Accuracy = acc_detect,
                                    Sensitivity = sen_detect,
                                    Specificity = spec_detect)

#Displaying the results
data.frame(results_detect) %>% knitr::kable()
```

Goal	Accuracy	Sensitivity	Specificity
anomaly Detection	0.9925	0.994	0.9909

We get an accuracy of 99.25%, Sensitivity of 99.4% and Specificity of 99.09% for Network anomaly Detection

7.2 Network Attack Classification

Using the Edx set to train the final model for Attack Classification and then making the predictions on the validation set:

```
edx_rf_attack_classify <- ranger(attack_class ~ ., data = edx.data_attack_class)

#Making predictions and getting Accuracy value
acc_classify <- confusionMatrix(predict(edx_rf_attack_classify, data = validation.data)$predictions,
                validation_set$attack_class)$overall["Accuracy"]

results_classify <- data_frame(Goal = "Attack Classification",
                                    Accuracy = acc_classify)
```

```
#Displaying the results
data.frame(results_classify) %>% knitr::kable()
```

Goal	Accuracy
Attack Classification	0.9926

We get an accuracy of 99.26% for Network Attack Classification.

8 Conclusion

Excellent accuracy values of over 99% have been achieved for both: Network anomaly Detection as well as Attack Classification. For the Binomial anomaly Detection, the Sensitivity and Specificity values are reasonably good at 99.4% and 99.1% respectivley.

A sensitivity of 99.4% implies that the rate of false alarms i.e the incorrect labeling of a normal network connection as one subject to an attack is only around 0.6%. High Sensitivity values are essential in a Network Intrussion Detection becuase blocking of benign network activity can severly impact productivity and business. High Specificity values are aslo highly desirable to maximise the detection of network attacks. A specificity of 99.1% implies that only around 0.9% of network attacks go undetecteded.

Although, substantial Dimension Reduction wasn't possible, we did manage to bring down the total number of predictors from 40 to 24 using Principal Component Analysis via Singular Value Decompoisiton of the Numerical Features. While PCA of mixed datatypes: Categorical and Numerical was attempted using the 'PCAmix' function under the 'PCAmixdata' package, we could have also performed Multiple Correspondance Analysis (MCA) on the categorical features to reduce the dimensions even more. MCA is an extension of the simple Correspondence Analysis, used during the Exploratory Data Analysis stages of this report, for summarizing and visualizing datasets containing multiple categorical variables. MCA can also be seen as a generalization of Principal Component Analysis when the variables to be analyzed are categorical instead of quantitative. The additional dimension reduction could have aided in reducing the computational time. Also, other algorithms such as KNN and GLM could have been used if the dimensions were reduced by a more sizeable amount.