A PREDICTION MODEL FOR FLOODED PACKETS IN SNMP NETWORKS

Aravindhan K[1], P Srikanth Kini[2] and G Adarsh Reddy[3]
School of Computing Science and Engineering, VIT University Chennai Campus, India
E-mail: {aravindhan.k, psrikanth.kini, gadarsh.reddy}2015@vit.ac.in

***Abstract*—This paper implements several machine learning algorithms to determine a model of maximum accuracy for predicting flooded packets in SNMP networks. Flooding is a simple routing technique in computer networks where a source or node sends packets through every outgoing link except for the source link. Flooding is also used as a denial of service(DOS) attack by flooding network traffic to bring down a network service. So, to determine the flooding of packets we implemented algorithms like SVM (Support Vector Machines), Logistic Regression, Decision-Tree Classifier, K-Neighbors Classifier, XGB Classifier and other algorithms and have compiled the results for each algorithm in terms of accuracy.**

***Keywords*—Flooding, SNMP packets, DOS, SVM, Logistic Regression, Decision-Tree Classifier, K-Neighbors Classifier, XBG Classifier**

1. INTRODUCTION

THE dataset used for the purpose of this project is that of SNMP networks, the cloud security dataset which is built using open source cloud. It is free software and you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. DDoS Attack datasets are generated using virtual instances running in a cloud security testbed. OpenStack private cloud is deployed in the cloud security testbed along with virtual instances running on them. The virtual instances are acting as zombies from different public and private networks of Open Stack private cloud. The attack generation is done using virtual instances as zombies and the data collection is performed at the victim with network monitoring software. The DDoS flood attacks that are considered in the experiment are ICMP flood, TCPSYN flood, TCPSYN-ACK flood, UDP flood and Land Flood. The attack duration is half an hour and the dump files collected are in the excel format.

The Simple Network Management Protocol (SNMP) is a common site in most networks. It not only provides a method that can be used for event notification but also can be used to maintain device statistics, set configuration parameters, and many other functions. There are many methods to detect flooding but this paper gives an overview of several algorithms implemented to achieve best results.

2. ARCHITECTURE DIAGRAM

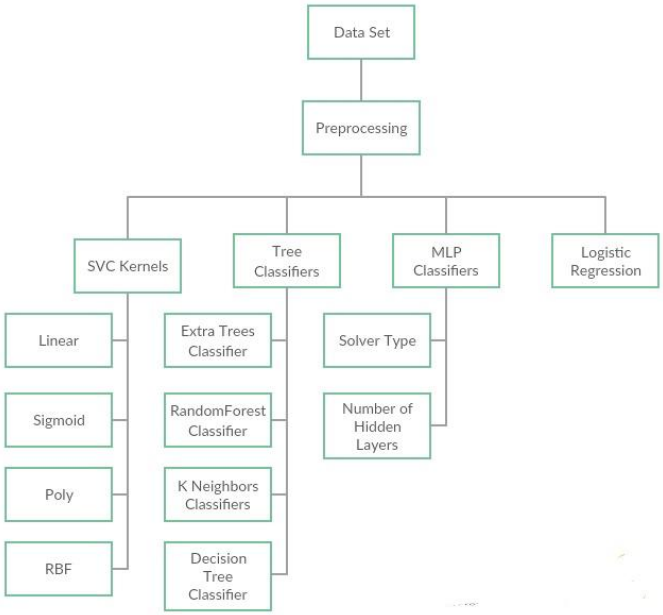

Fig. 1: Architecture Diagram

3. RELATED WORKS

SNMP provides a universal method of exchanging data for purposes of monitoring systems that reside on a network. But, to utilize SNMP for traffic flooding attack detection we need to consider the following three points in the use of the SNMP MIB objects which affects the performance and accuracy of the detection system:

1) Proper selection of SNMP MIB objects for attack detection,
2) Determination of the detection timing about when and how often,
3) Algorithm for attack detection using the selected MIB objects.

Attack detection methods employed for flooding include Support Vector Machines using testbed network architecture and Yoo et al. utilized tcpInErrs, udpNoPorts, and icmpOutEchoReps SNMP MIB objects for the purpose of detecting DoS/DDoS attacks. These MIB objects are not suitable to detect certain type of modern attacks.

3.1 Testbed network structure

Jaehak Yu et al. constructed a testbed network to carry out an actual attack experiment. The network topology consisted of one victim system, two attack agent systems, one attack handler system, and one dataset collector system. The OS of the victim system is Linux Fedora 7. Linux Fedora 8 is used for the OS of other systems. Several servers (Web server, SSH/SFTP server, VNC server, Samba server, and MRTG server, etc.) are working on the victim system. The testbed network is connected to the campus network, so normal traffic is generated between the victim host and other hosts outside the testbed network during the experiment period.

They used Stacheldraht, a distributed denial of service attack tool, to generate attack traffic. The Stacheldraht was selected because it is a more mature attack tool compared to other attack tools, such as TFN, TFN2K, or Trinoo. The Stacheldraht is composed of handler (master) and agent (daemon) programs. The handler system scans vulnerabilities of the victim host before sending an attack command to the corresponding multiple agent systems.

Agent systems produce a large flood of packets targeting the victim host, which sabotage both systems resources and network resources. For the experiments we conducted three types of network flooding attacks: TCP-SYN flooding, UDP flooding, and ICMP flooding attacks.

3.2 SNMP MIB variables

During the attack experiment, the dataset collector system gathered SNMP MIB data from the victim system using SNMP query messages. Firstly, we investigated 66 MIB variables from five MIB-II groups: Interface, IP, TCP, UDP, and ICMP. The data types of the 66 MIB variables are Counter, which is a non-negative 4-byte integer that may be incremented but not decremented. These MIB variables are continuously updated as the outgoing and incoming traffic from/to a system occur, which could possibly be used for attack detection.

They selected 13 MIB variables among 66 MIB variables, which are likely to be affected by the attack traffic by a comprehensive but not exhaustive investigation. Most of the 13 MIB variables are used in for traffic flooding attack detection. In the first phase of their experiment, they used these 13 SNMP MIB variables for the proposed detection mechanism. The 13 MIB variables and their corresponding MIB groups are shown in Table 1. In the second phase of their experiment, they selected 3 MIB variables and 5 MIB variables among the 13 MIB variables for each stage of the attack detection mechanism by the aforementioned feature selection method CFS. The first 3 MIB variables are used for the first-level attack determination from normal traffic. The next 5 MIB variables are used for the second-level attack type classification among attack traffic. The udpInError MIB variable was selected in both levels by CFS. Only 7 distinct MIB variables among 13 MIB variables are used in the second phase of the detection mechanism.

MIB Group	SNMP MIB Variables
IP	ip.ipInReceives (3)
	ip.ipInDelivers (9)
	ip.ipOutRequests (10)
	ip.ipOutDiscards (3)
TCP	Tcp.tcpAttemptsFails (7)
	Tcp.tcpOutRsts (7)
UDP	Udp.udpInErrors (3)
ICMP	Icmp.icmpInMsgs (1)
	Icmp.icmpInErrors (2)
	Icmp.icmpInDestUnreachs (2)
	Icmp.icmpOutMsgs (14)
	Icmp.icmpInErrors (15)
	Icmp.icmpOutDestUnreachs(16)

Table 1: The SNMP MIB variables used for the attack detection Mechanism

3.3 *Attack identification result*

Firstly, they performed an identification test of the proposed mechanism between attack traffic and normal traffic. They performed a SVDD training with 1000 normal MIB records of a training dataset, then, tested with 2500 MIB records of a testing dataset: 1000 normal MIB records and three 500 MIB records for three attacks types: TCP-SYN, UDP, and ICMP flooding attacks. The MIB records in each dataset are selected randomly among all 5000 MIB records. Three important formulas were used to evaluate the performance of the first step of identification: the attack detection rate, false positive rate (FPR), and false negative rate (FNR), which are as follows:

$$\text{Attack Detection Rate} = \frac{\sum_{i=1}^{n} T_i}{\sum_{i=1}^{n} I_i}$$

$$\text{False Positive Rate} = \frac{\sum_{i=1}^{n} P_i}{\sum_{i=1}^{n} N_i}$$

$$\text{False Negative Rate} = \frac{\sum_{i=1}^{n} F_i}{\sum_{i=1}^{n} I_i}$$

In the above equation, I is an individual attack traffic MIB record, while N is a MIB record for normal traffic. In our experiment, the number of attack traffic records and normal traffic records are 1500 and 1000, respectively. T is an attack traffic record which is classified as an attack by the system. P indicates a normal traffic record which is misclassified as attack traffic. F is an attack traffic record which is misclassified as

normal traffic.
The overall attack detection rates of the two phases are 97.07% and 99.40%, respectively. The second phase with CFS-selected MIB variables outperforms the first phases; this means that the attack identification is highly affected by the selection of MIB variables as well as the attack detection mechanism, and CFS is a good choice for the MIB variable selection. The FPR is the rate of misclassified normal traffic, as attack traffic over total normal traffic. The FNR is the rate of misclassified attack traffic, as normal traffic over total attack traffic. It is generally accepted that the FNR is more important than the FPR for evaluation of the performance of an intrusion detection system.

4. PROPOSED METHOD

The dataset used for the project contains several attributes which have been elaborated in the next section.

4.1 Correlation between the attributes

The relationship between the attributes was studied by plotting a heat-map or correlation map which essentially showed how attributes with similar values are clustered. The default color gradient sets the lowest value in the heat map to dark blue, the highest value to a bright red, and mid-range values to light gray, with a corresponding transition (or gradient) between these extremes. (Refer Appendix1Figure 1)

4.2 Pre-processing of the data

4.2.1 Dropping of columns

There were several data columns with similar values which in terms of analysis have no significance. Redundant columns with repeating insignificant values were eliminated to create a reduced data set. Out of the 66 columns we had, 36 were dropped. So a total of 30 columns were left to work with. The class labels were replaced with integers ranging from 0-9 with the different types of floods receiving the latter set of values.

4.2.2 Making frames

Since it is not possible to determine if a single packet is flooded, it was necessary to group packets into frames for analysis. We created a function TimeDiff() using Python which takes the data column and splits the attribute in terms of hours, minutes and seconds to find the difference between consecutive packets sent. We grouped 5 packets at a time and calculated the difference in their arrival time. This function precisely calculates time difference between consecutive packets and creates a separate data set with the updated values. The packet values are considered as a single data frame or point. This is used to reduce the overall dataset and find the time difference between packets.

4.3 Training and test sets

The reduced dataset was used to split into training and test data sets. 70% of the dataset was used for training and 30% was used for testing which were subjected to several classification algorithms and tested for accuracy. Before classification we used the Standardscaler () function in the sklearn preprocessing library, to scale the dataset across the attributes.

5. CLASSIFIERS USED

Under the sklearn library, several classifier algorithms used are compiled below:

5.1 SVC

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. We used the SVC classifier on the dataset with the linear, sigmoid, poly and RBF kernels.

5.2 MLPClassifier

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target y, it can learn a non-linear function approximate for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output.

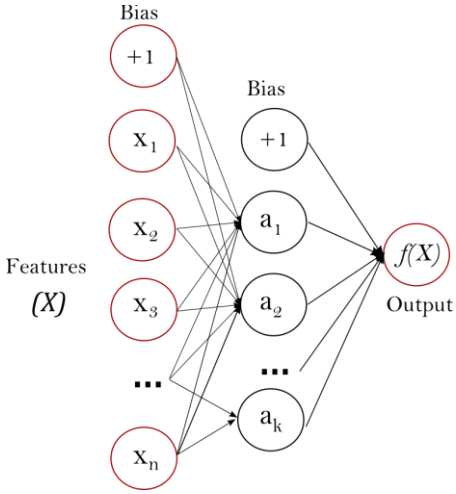

Fig. 2: *One hidden layer MLP*

5.3 Logistic Regression

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag' and 'lbfgs' solvers. It can handle both dense and sparse input. It uses C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted.

5.4 DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

5.5 KNeighborsClassifier

NearestNeighbors implements unsupervised nearest neighbors learning. It acts as a uniform interface to three different nearest neighbors algorithms: **BallTree**, **KDTree**, and a brute-force algorithm based on routines in **sklearn.metrics.pairwise**. The choice of neighbors search algorithm is controlled through the keyword 'algorithm', which must be one of ['auto', 'ball_tree', 'kd_tree', 'brute']. When the default value 'auto' is passed, the algorithm attempts to determine the best approach from the training data.

5.6 RandomForestClassifier

The **sklearn.ensemble** module includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-Trees method. Both algorithms are perturb-and-combine techniques specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

5.7 XGBClassifier

XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework. The XGBoost model for classification is called **XGBClassifier**. We can create and and fit it to our training dataset. By default, the predictions made by XGBoost are probabilities. Because this is a binary classification problem, each prediction is the probability of the input pattern belonging to the first class. We can easily convert them to binary class values by rounding them to 0 or 1.

5.8 ExtraTreesClassifier

An extra-trees classifier.
This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

5.9 GradientBoostingClassifier

Gradient Boosting for classification. GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

6. DATA USED

The SNMP parameters that are used for data collection are as follows:
SNMP Parameters:
These parameters include all the protocols like IP, ICMP, UDP and TCP OID's where the following metrics can be obtained:

ipForwarding:
The indication of whether this entity is acting as an IP router in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP routers forward datagrams. IP hosts do not (except those source-routed via the host).

ipDefaultTTL:
The default value inserted into the Time-To- Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

ipInReceives:
The total number of input datagrams received from interfaces, including those received in error.

ipInHdrErrors:
The number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to- live exceeded, errors discovered in processing their IP options, etc

ipInAddrErrors:
The number of input datagrams discarded because the IP address in their IP headers destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP routers and

therefore, do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address.

ipForwDatagrams:
The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP routers, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful

ipInUnknownProtos:
The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol. cause of an unknown or unsupported protocol.

ipInDiscards:
The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly.

ipInDelivers:
The total number of input datagrams successfully delivered to IP user-protocols (including ICMP).

ipOutRequests:
The total number of IP datagrams which local IP user- protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams.

ipOutDiscards:
The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion.

ipOutNoRoutes:
The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this `no-route criterion. Note that this includes any datagrams which a host cannot route because all of its default routers are down.

ipReasmTimeout:
The maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity

ip_ReasmReqds:
Number of Internet Protocol (IP) fragments received which needed to be reassembled at this entity.

ipReasmOKs:
Number of Internet Protocol (IP) datagrams successfully re-assembled.

ip_ReasmFails:
Number of failures detected by the IP reassembly algorithm (for whatever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received.

ip_FragOKs:
Number of Internet Protocol (IP) datagrams that have been successfully fragmented at this entity.

ip_FragFails:
The number of Internet Protocol (IP) datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g. because their Dont Fragment flag was set.

ip_FragCreates:
Number of Internet Protocol (IP) datagram fragments that have been generated as a result of fragmentation at this entity.

ip_RoutingDiscards:
Number of routing entries which were chosen to be discarded even though they are valid.

tcp_RtoAlgorithm: Algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

tcp_RtoMin:
The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre, an object of this type has the semantics of the LBOUND quantity described in RFC 793.

tcp_RtoMax:
The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre, an object of this type has the semantics of the UBOUND quantity described in RFC 793.

tcp_MaxConn:
The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value -1.

tcp_ActiveOpens:
The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.

tcp_PassiveOpens:
The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.

tcp_AttemptFails:
The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.

tcp_EstabResets:
The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.

tcp_CurrEstab:
The number of TCP connections for which the current state is either ESTABLISHED or CLOSEWAIT.

tcp_InSegs:
The total number of segments received, including those received in error. This count includes segments received on currently established connections.

tcp_OutSegs:
The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets.

tcp_RetransSegs:
The total number of segments retransmitted – that is, the number of TCP segments transmitted containing one or more previously transmitted octets.

tcp_InErrs:
The total number of segments received in error (e.g., bad TCP checksums).

tcp_OutRsts:
The number of TCP segments sent containing the RST flag.

udp_InDatagrams:
The total number of UDP datagrams delivered to UDP users.

udp_NoPorts:
The total number of received UDP datagrams for which there was no application at the destination port.

udp_InErrors:
The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

udp_OutDatagrams:
The total number of UDP datagrams sent from this entity.

icmp_InMsgs:
The total number of ICMP messages which the entity received. Note that this counter includes all those counted by icmpInErrors.

icmp_InErrors:
The number of ICMP messages which the entity received but determined as having ICMP-specific errors (bad ICMP checksums, bad length, etc.

icmp_inDestUnreachs:
The number of ICMP Destination Unreachable messages received.

icmp_InTimeExcds:
The number of ICMP Time Exceeded messages received.

icmp_InParmProbs:
The number of ICMP Parameter Problem messages received.

icmp_InSrcQuenchs:
The number of ICMP Source Quench messages received.

icmp_InRedirects:
The number of ICMP Redirect messages received.

icmp_InEchos:
The number of ICMP Echo (request) messages received.

icmp_InEchoReps:
The number of ICMP Echo Reply messages received.

icmp_InTimestamps:
The number of ICMP Timestamp (request) messages received.

icmp_InTimestampReps:
The number of ICMP Timestamp Reply messages received.

icmp_InAddrMasks:
The number of ICMP Address Mask Request messages received.

icmp_InAddrMaskReps:
The number of ICMP Address Mask Reply messages received.

icmp_OutMsgs:
The total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors.

icmp_OutErrors:
Number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counters value
.

icmp_OutDestUnreachs:
The number of ICMP Destination Unreachable messages sent.

icmp_OutTimeExcds:
The number of ICMP Time Exceeded messages sent.

icmp_OutParmProbs:
The number of ICMP Parameter Problem messages sent.

icmp_OutSrcQuenchs: The number of ICMP Source Quench messages sent.

icmp_OutRedirects:
The number of ICMP Redirect messages sent. For a host, this object will always be zero since hosts do not send redirects.

icmp_OutEchos:
The number of ICMP Echo (request) messages sent.

icmp_OutEchoReps:
The number of ICMP Echo Reply messages sent.

icmp_OutTimestamps: The number of ICMP Timestamp (request) messages sent
icmp_OutTimestampReps:
The number of ICMP Timestamp Reply messages sent.

icmp_OutAddrMasks:
The number of ICMP Address Mask Request messages sent.

icmp_OutAddrMaskReps:
The number of ICMP Address Mask Reply messages sent.

7. COMPARATIVE STUDY BETWEEN DIFFERENT ALGORITHMS

The below comparison in Fig 3 is between the different types of SVC Kernels of which best accuracy was given by the Linear SVC Kernel

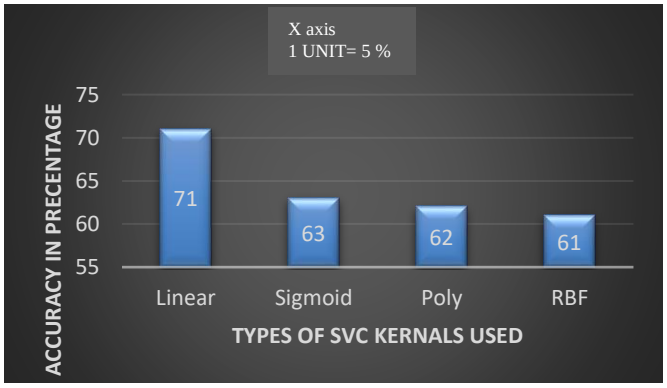

Fig. 3: Comparison Between Different SVC Kernels

The below comparison in Fig 4 is between the different types of Tree Classifier Algorithms of which best accuracy was given by the Extra Trees Class

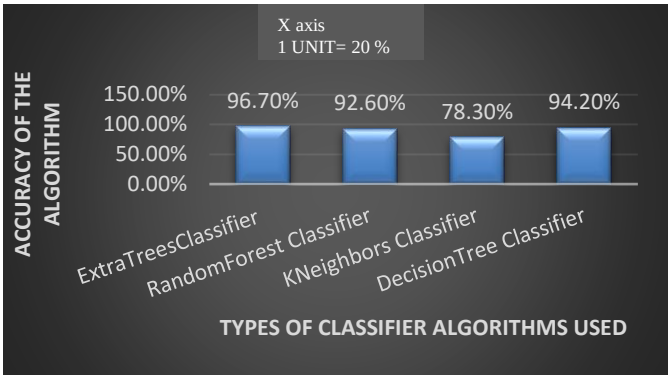

Fig. 4: Comparison Between Different Tree Classifiers

The below comparison in Fig 5 is between MLP Classifier with different parameters of which best accuracy was given by the LBFGS Solver with 50 Hidden Layers

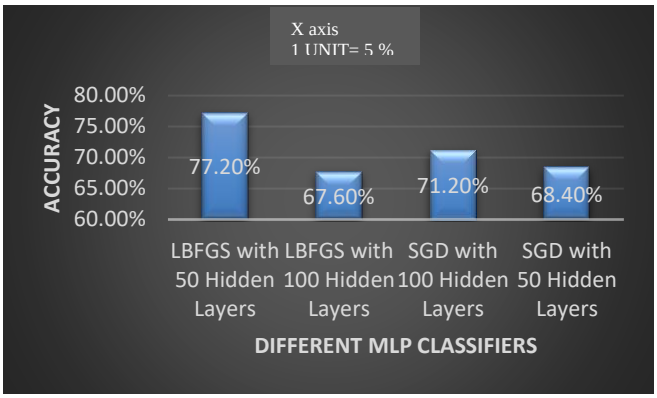

Fig. 5: Comparison Between MLP Classifiers with Different Number of hidden layers

8. RESULTS

The objective of the experiment conducted was to measure the accuracy difference between various classifiers for the dataset. Hence the focus was not in improving the accuracy of individual classifiers but to compare the various classifier algorithms. In order to compare the performance of the classifiers, the accuracies were measured for individual classifiers. The measured accuracies for individual classifiers are displayed in the Fig 6.

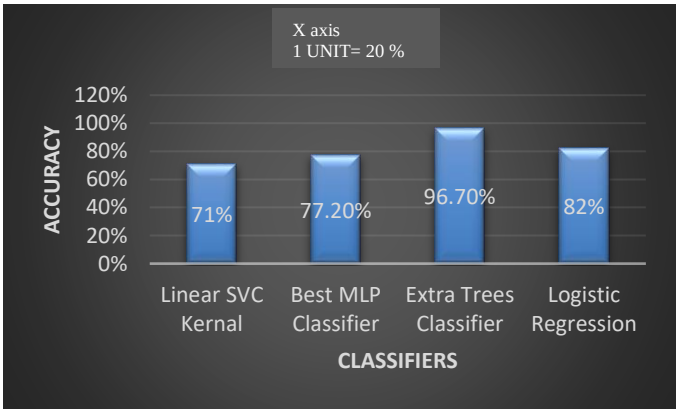

Fig. 6: Comparison Between Best Algorithms in their classifications

9. REFERENCES

[1] P. Suresh Kumar,S, Pranavi, “Performance analysis of machine learning algorithms on diabetes dataset using big data analytics”

[2] M. Kim, H. Kang, S. Hong, S. Chung, J.W. Hong, A flow -based method for abnormal network traffic detection, in: Proceedings of NOMS 2004, Seoul, Korea, April 2004 , pp. 559–612

[3] Cui-Mei Bao,”Intrusion detection based on one-class svm and snmp mib data”

[4] https://www.python-course.eu/machine_learning.php

[5] https://www.datacamp.com/community/tutorials/pyth on-numpy-tutorial

[6]Jianwu Zhang,” Design and implementation of test IP network intelligent monitoring system based on SNMP”

[7] https://www.datacamp.com/community/tutorials/pan das-tutorial-dataframe-python

[8] Marina K. Thottan, George K. Swanson, Michael Cantone,”SEQUIN: An SNMP-based MPLS network monitoring system”

[9] https://www.analyticsvidhya.com/blog/2016/01/com plete-tutorial-learn-data-science-python-scratch-2/

[10] Lei Li, Yabin Wu, Yihang Ou,”Research on machine learning algorithms and feature extraction for time series”

[11] Heng Zhou, Guangpu Shan, Song Liu,” A Monitoring Method of Network Power Consumption Information Based on SNMP”

[12] https://www.geeksforgeeks.org/numpy-in-python-s et-1-introduction/

[13] Achmad Affandi, Dhany Riyanto , Istas Pratomo ,“Design and implementation fast response system monitoring server using Simple Network Management Protocol”

[`4] https://www.hackerearth.com/practice/machine-lear ning/data-manipulation-visualisation-r-python/tutor ial-data-manipulation-numpy-pandas-python/tutorial/

[15] L.P. Gaspary, R.N. Sanchez, D.W. Antunes,” A SNMP-based platform for distributed stateful intrusion detection in enterprise networks”

[16] Yoo, D.-S., Oh, C.-S.: Traffic Gathering and Analysis Algorithm for Attack Detection. In: KoCon 2004 Spring Integrated conference, vol. 4, pp. 33–43 (2004)

[17] Kittikhun Thongkanchorn, Sudsanguan Ngamsuriyaroj, Vasaka Visoottiviseth, “Evaluation studies of three intrusion detection systems under various attacks and rule sets”

[18] Hao Wei, Micheal Baechler, Fouad Slimane, “Evaluation of SVM, MLP and GMM Classifiers for Layout Analysis of Historical Documents”

[19] Xuan Zhou, Wenjun Wu, Yong Han,” Modeling multiple subskills by extending knowledge tracing model using logistic regression”

[20] Chris Yakopcic, Tarek M. Taha, “Memristor crossbar based implementation of a multilayer perceptron”

10. Appendix

1)**Heat-Map**

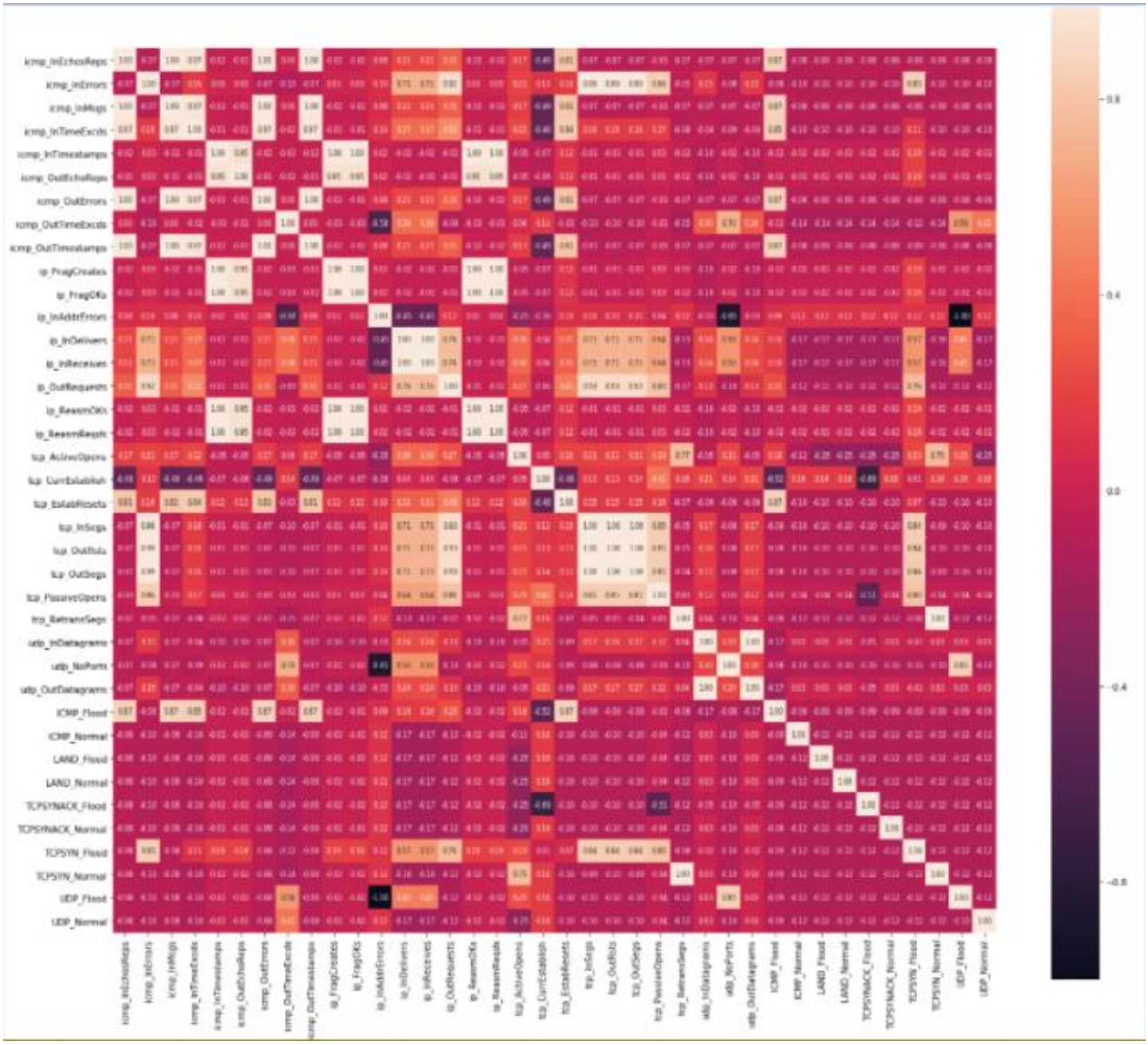

Fig. 7: HeatMap