Artificially Intelligent Intrusion Detection System

Charles Rizzo
University of Tennessee
crizzo@vols.utk.edu

Nick Skuda
University of Tennessee
nskuda@vols.utk.edu

Austin Saporito
University of Tennessee
dxh594@vols.utk.edu

***Abstract*—Intrusion Detection Systems are software systems that monitor network traffic for malicious activity. Complex systems are multifaceted and have functionalities like deep packet inspection, breach detection, and anomaly detection. The systems respond to suspicious activity by causing alerts and notifying network administrators of the offending behavior.**

Machine learning methodologies have been studied and developed to perform tasks like classification, regression, and even event detection. This paper seeks to apply several different machine learning methodologies to the UNSW-NB15 network intrusion detection data set. By applying and comparing several different models to the data set, the goal is to maximize malicious event classification while minimizing false alarms and determine which models best differentiate between malicious and normal behavior.

***Index Terms*—UNSW-NB15, Intrusion detection system, machine learning**

I. Introduction

The 1998 and 1999 DARPA Intrusion Detection Data Sets are several weeks worth of raw network TCP data that was collected and organized by Lincoln Laboratory MIT. This data was used to study and develop intrusion detection systems that were subsequently delivered to the Air Force Research Laboratory for real-time evaluation [1]. These data sets were considered the standards for developing network intrusion detection systems for over two decades. However, they are indeed two decades outdated and needed to be updated. To accomplish this, the UNSW-NB15 data set was created in 2015 as a hybrid of real modern normal and the contemporary synthesized attack activities of the network traffic [2].

The raw data was about 100 gigabytes of network traffic as pcap files that yielded 2,540,044 connection records. The data was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security. The Argus and Bro-IDS tools were used along with 12 algorithms to transform the the data into connection records comprised of a set of 49 features with label. Each connection record is defined as a sequence of TCP packets that have well-defined starting and ending times as well as discrete source and destination IP addresses. Each connection record is labeled either as normal or as one of 9 types of attacks. [2].

The attacks are labeled according as one of the nine following types: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. In the appendix are Tables I, II, III, IV, V, VI, and VII that list and detail each of the 49 features that comprises a connection record [2]. The facts that the UNSW-NB15 data set has more features per connection record and includes real modern normal attack activities are two main motivations for the usage of this data set compared to the DARPA-derived KDD-Cup data set.

The conversion of the raw TCP data into connection records facilitates the use of machine learning classification techniques to be fit to the data and make predictions as to what kind of attack, if any, a test data set connection record is. This work aims to evaluate the performance of several machine learning models provided by Scikit-Learn in order to determine how certain techniques do across the entire data set. Because two of the authors (Nick and Charlie) are members of the TENNLab Neuromorphic group, this work also leverages and compares the performance of recurrent, spiking neural networks against classical machine learning algorithms. The motivation for doing this is that recurrent, spiking neural networks are typically smaller and therefore more lightweight than conventional neural networks (in the form of Multi-layer perceptrons). Their lightweightedness contributes to their ability to function and produce classifications in real-time, an important faculty for a network intrusion detection system to possess.

II. Methodology

Since the authors provided both the classification labels (one of the aforementioned nine attacks and ”Normal”) as well as binary labels (0 for ”Normal” and 1 for one of the nine attacks), we decided to work four versions of the data set to see how performance was impacted by classification scheme and whether or not the feature space was reduced with PCA. Plainly, the four data sets were the following: binary classification without PCA, binary classification with PCA, labelled classification with PCA, and labelled classification without PCA.

For the Principal Component Analysis we opted to retain 90% of the variance in the principal components (or introduce a 10% error) which reduced the feature space from 49 features to 16 (shown in Figure 1), resulting in models developing more quickly. The trade-off with reducing the amount of features is typically accuracy for training and prediction time. We reduced the amount of features across the board as opposed to hand tuning and selecting certain features.

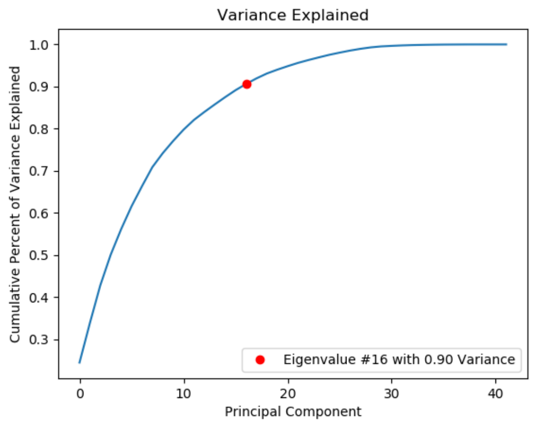

Fig. 1: Amount of Principal Components Used to Retain 90% Variance

We used only supervised machine learning techniques through Scikit-Learn. This decision to not use unsupervised techniques is due to the fact that this is a labeled data set; therefore, it doesn't make sense to us to use an unsupervised machine learning technique for this task. The supervised machine learning models that we trained and developed were the following: k-nearest neighbors, backpropagation neural networks (multilayer perceptrons), random forest, support vector machines (SGD Classifier), and Naive Bayes.

Neuromorphic systems classify using a neural network similarly to traditional deep neural networks [3]. However unlike feed-forward artificial networks, neuromorphic systems, or spiking neural networks (SNNs) have a temporal aspect to classification, where spikes flow over a period of simulated time, in response to the set of inputs specified. This temporal aspect may be useful for tasks which have a temporal component, which includes traffic analysis.

We developed each model to distinguish between all of the different attacks as as well as a binary scheme where the models merely predict either "normal" or "malicious" behavior. Ultimately, a binary classification scheme is a more realistic approach because in the real world, a model would never be able to correctly label an attack by name if it had not been trained to recognize that particular attack. However, it is a realistic expectation for a model to be able to differentiate between normal and malicious behavior based on prior known (and trained on) malicious behavior or attacks.

For evaluation, we generated training and testing F1-scores for each model across the entire data set. The reason we are going to report F1-scores instead of raw accuracy is because there is an imbalance in the distribution of events in the data set. Reporting F1-scores will take into account the difference in different label weights relative to how many samples it occupies in the data set. As a function of the label distributions shown in Figures 2 and 3, we also decided to weigh the "Normal" event samples roughly 5 times that of an attack sample. This was done to help encourage a lower false positive rate. Just like normal accuracy, however, the best F1-score a model can achieve is 1. While developing the models, we will use 10-fold cross validation to ensure that our best generalizing models' parameter sets are the ones that are reported as being the best instead of merely reporting the parameter combinations with the best performances on isolated instances of the data set. We will also generate confusion matrices for each model and a Receiver Operating Characteristic (ROC) graph to detail each models' separability performance overall. We will also report and highlight both the performance (or true detection) and false alarm/false positive rates per model for the binary classification experiment.

To train a neuromorphic system, we used the TENNLab neuromorphic research framework [4], with the EONS [5] training technique. The TENNLab framework allows us to define the problem and dataset and all associated hyperparameters. This research framework is in active development, and currently has limited support for classification tasks, and as such the TENNLab framework does not have support for differential weights for classes during training. As such we trained without adjusted weights, which may not give a direct comparison but may establish if this training technique is viable for this kind of problem. Due to the long training times, the neuromorphic system was trained without cross-validation.

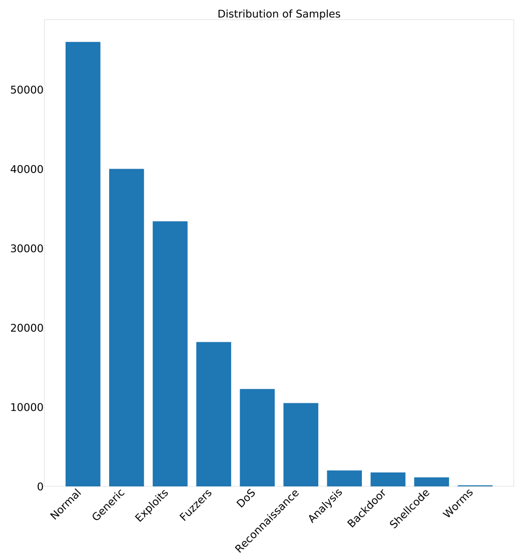

Fig. 2: Sample Distribution in Training Set

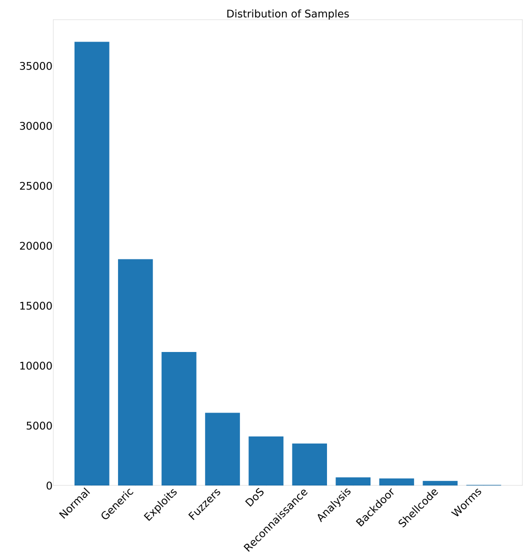

Fig. 3: Sample Distribution in Testing Set

A known issue in machine learning is the problem of hyper-parameter selection. The values of some models' parameters are continuous, which means there are an infinite number of ways that a particular model can be created and trained around the data set. In order to address this, we will define a static list of parameter values that each of the models' could take and randomly select 100 combinations of these parameters to test and generate models from. This random search approach is shown to be about as effective as an exhaustive grid search but saves much in the way of time and computational resources [6].

Hardware-wise, each of the models will be trained and developed on the Neuro-Firewall computer cluster at the University of Tennessee at Knoxville. It is a 1728 core system comprising 36 Dell PowerEdge C6145s. Each node features quad 12 core 64 bit AMD Opteron Processor 6180 SEs and 96 GBs of RAM. In terms of software, all of the code will most likely be written in and run with Python 3.7.4, and all of the models and evaluation metrics will come from the Scikit-Learn 0.22.1 library [7].

III. Results

For simplicity, Table I shows a condensed version of our training and testing F1-Scores as well as prediction time for the scikit-learn models.

	Train F1-Score	**Test F1-Score**	**Prediction Time (s)**
kNN Labels	0.8358	0.7739	446.78
kNN Labels PCA	0.8290	0.5974	101.18
kNN Binary	0.9284	0.7940	376.76
kNN Binary PCA	0.9233	0.6302	881.70
MLP Labels	0.8315	0.7869	0.52
MLP Labels PCA	0.8227	0.5819	0.50
MLP Binary	0.9179	0.8067	0.38
MLP Binary PCA	0.9033	0.6959	0.07
SGD Classifier Labels	0.7831	0.7948	0.03
SGD Classifier Labels PCA	0.7204	0.6017	0.03
SGD Classifier Binary	0.8754	0.6971	0.01
SGD Classifier Binary PCA	0.8489	0.6072	0.006
Random Forest Labels	0.8694	0.7039	1.16
Random Forest Labels PCA	0.8391	0.5986	4.52
Random Forest Binary	0.9466	0.6655	0.72
Random Forest Binary PCA	0.9246	0.6670	0.16
Naive Bayes Labels	0.5649	0.6917	0.47
Naive Bayes Labels PCA	0.6514	0.4788	0.23
Naive Bayes Binary	0.8290	0.6918	0.10
Naive Bayes Binary PCA	0.7661	0.4190	0.05
Neuromorphic Labels	0.5570	0.6148	12.27
Neuromorphic Labels PCA	0.6446	0.0002	3.42
Neuromorphic Binary	0.9497	0.8149	12.30
Neuromorphic Binary PCA	0.8511	0.7087	3.26

TABLE I: Classification Results

By and large, PCA resulted in quicker prediction times across the board, and, as expected, lower F1-scores. There were also a few instances where the testing F1-Score was greater than the training F1-Score, which is interesting and indicates that the model generalized pretty well. Random Forest trained the best on the binary data, but the Neuromorphic classifier actually generalized the best on the binary test data set.

Table II lists all of the False Positive and True Detection rates. A false positive occurs when a classifier predicts a sample as an attack when it was actually normal and a "true detection" is when a classifier correctly guesses an attack as an attack. The table shows that the model with the lowest false positive rate is (with a true detection rate that is nonzero) is the Neuromorphic classifier on the labeled data set. While having a 1% or less false positive rate is great, having only a true detection rate of 44% isn't great. Alternatively, the model with the highest true detection rate that doesn't always guess "attack" is the SGD Classifier on the binary data set. These numbers are better expressed in the ROC curves that follow (with the exception of Neuromorphic).

	False Positive Rate	True Detection Rate
kNN Labels	0.0315	0.6938
kNN Labels PCA	0.1048	0.5242
kNN Binary	0.2725	0.9473
kNN Binary PCA	0.4887	0.9138
MLP Labels	0.0161	0.6197
MLP Labels PCA	0.0792	0.5187
MLP Binary	0.3531	0.9871
MLP Binary PCA	0.4042	0.9451
SGD Classifier Labels	0.1363	0.8290
SGD Classifier Labels PCA	0.4858	0.6163
SGD Classifier Binary	0.4222	0.9988
SGD Classifier Binary PCA	0.5191	0.9284
Random Forest Labels	0.0147	0.5906
Random Forest Labels PCA	0.0692	0.5223
Random Forest Binary	0.4189	0.9973
Random Forest Binary PCA	0.4778	0.8910
Naive Bayes Labels	0.9995	1.0
Naive Bayes Labels PCA	0.0929	0.3546
Naive Bayes Binary	0.0	0.0005
Naive Bayes Binary PCA	0.7289	0.9973
Neuromorphic Labels	0.0108	0.4478
Neuromorphic Labels PCA	0.0026	0.0000
Neuromorphic Binary	0.4684	0.9511
Neuromorphic Binary PCA	0.9898	0.9934

TABLE II: False Positive and True Detection Rates

The following four ROC curve graphs (one per data set) illustrates the models' distiguishability or separability. Figures 4, 5, 6, and 7 show the relationship between the models' specificity and sensitivity: in short, the more to the upper left a curve is, the better that model is at distinguishing labels on that particular data set.

Our highest Area Under the Curve (AUC) score is seen in Figure 6. MLP is able to best distinguish between "Normal" and "Attack" behavior. Looking at the rates in Table 2 reinforces this. It's not surprising that in general, the ROC AUCs are higher for the binary data sets as opposed to the data sets that are comprised of 10 labels to distinguish between. An interesting note is how in Figure 4, the Naive-Bayes classifier is no better at distinguishing between the ten labels than a uniformly random classifier (that's what the dashed line symbolizes – AUC = 0.5). This is also seen in Figure 5 with the kNN classifier. In general, PCA resulted in lesser AUCs, which is not surprising.

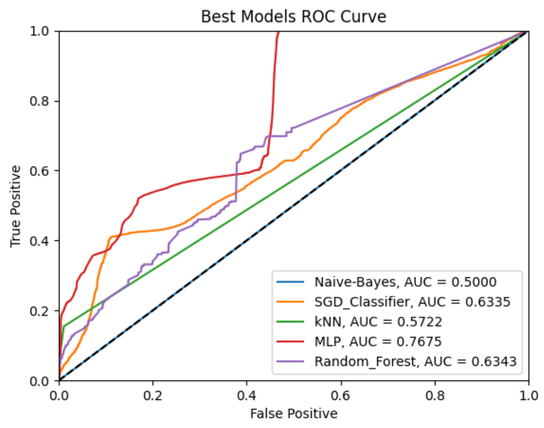

Fig. 4: ROC Curves for Labeled Classification

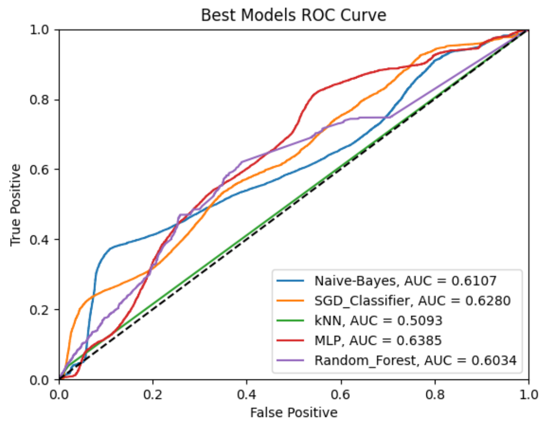

Fig. 5: ROC Curves for PCA Labeled Classification

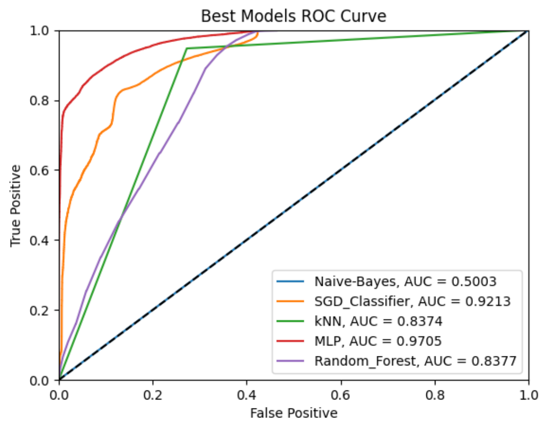

Fig. 6: ROC Curves for Binary Classification

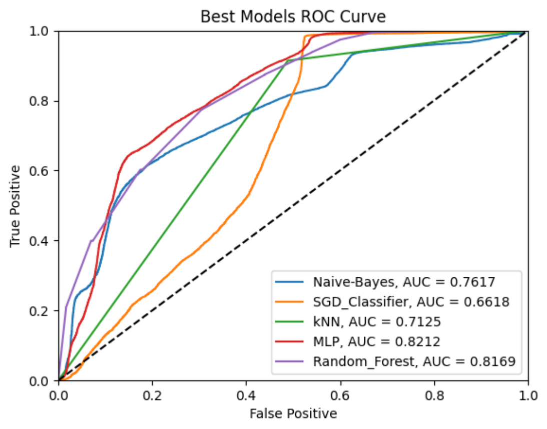

Fig. 7: ROC Curves for PCA Binary Classification

IV. Future Work and Issues

Altogether, there were not many issues that occurred over the course of this project. There was some difficulty in successfully getting the experimental neuromorphic framework working with the data set on the neuro-firewall cluster, but that was resolved pretty quickly. The group's experience with machine learning facilitated a pretty smooth workflow and separation of responsibilities.

In terms of future work, there are several avenues that could be explored. With the data set, it might be worth trying to weigh the ”normal” samples differently to analyze which sample weight yields the lowest false positive rates across the board and how they affect the true detection rates.

Given the research-esque nature of the neuromorphic workflow, there is a lot of room to try different learning algorithms, different spike encoder/decoders, different neuroprocessors, etc. However, trying different combinations and parameters is a problem that grows exponentially. Specifically within the framework, it'd be interesting to try using a Bayesian optimization utility to see how well it helps tune some of the neuromorphic components' parameters.

V. Timeline and Responsibilities

Timeline-wise, we had the scikit-learn models trained and developed by the end of October or early November. Once the the cluster was done developing the scikit-learn models, Nick took over using it to develop the neuromorphic solution. We had most of November to generate the graphs, tables, and figures shown throughout the paper.

Austin and Charlie split the workload of Scikit-Learn classifiers and graph generation and worked more closely together in doing so. Nick handled all of the neuromorphic classification work; because of how much more complicated it was, he spent many hours configuring a classification application that works with the neuromorphic framework, selecting a neuroprocessor and its parameters, choosing input/output encoding schemes, etc.

References

[1] “1998 darpa intrusion detection evaluation dataset.”

[2] N. Moustafa and J. Slay, “Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set),” *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015.

[3] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, “A survey of neuromorphic computing and neural networks in hardware.” arXiv:1705.06963, May 2017.

[4] J. S. Plank, C. Rizzo, K. Shahat, G. Bruer, T. Dixon, M. Goin, G. Zhao, J. Anantharaj, C. D. Schuman, M. E. Dean, G. S. Rose, N. C. Cady, and J. Van Nostrand, “The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems,” in *44th Annual GOMACTech Conference*, (Albuquerque), March 2019.

[5] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, “Evolutionary optimization for neuromorphic systems,” in *NICE: Neuro-Inspired Computational Elements Workshop*, 2020.

[6] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix

	Feature	**Type**	**Description**
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol

TABLE III: Flow Features

	Feature	**Type**	**Description**
6	state	nominal	The state and its dependent protocol, e.g. ACC, CLO, else (-)
7	dur	float	Record total duration
8	sbytes	integer	Source to destination bytes
9	dbytes	integer	Destination to source bytes
10	sttl	integer	Source to destination time to live
11	dttl	integer	Destination to source time to live
12	sloss	integer	Source packets retransmitted or dropped
13	dloss	integer	Destination packets retransmitted or dropped
14	service	nominal	http, ftp, ssh, dns ..,else (-)
15	sload	float	Source bits per second
16	dload	float	Destination bits per second
17	spkts	integer	Source to destination packet count
18	dpkts	integer	Destination to source packet count

TABLE IV: Basic Features

	Feature	**Type**	**Description**
19	swin	integer	Source TCP window advertisement
20	dwin	integer	Destination TCP window advertisement
21	stcpb	integer	Source TCP sequence number
22	dtcpb	integer	Destination TCP sequence number
23	smeansz	integer	Mean of the flow packet size transmitted by the src
24	dmeansz	integer	Mean of the flow packet size transmitted by the dst
25	trnas_depth	integer	the depth into the connection of http request/response transaction
26	res_bdy_len	integer	The content size of the data transferred from the server's http service.

TABLE V: Content Features

	Feature	**Type**	**Description**
27	sjit	float	Source jitter (mSec)
28	dhit	float	Destination jitter (mSec)
29	stime	timestamp	record start time
30	ltime	timestamp	record last time
31	sintpkt	float	Source inter-packet arrival time (mSec)
32	dintpkt	float	Destination inter-packet arrival time (mSec)
33	tcprtt	float	The sum of 'synack' and 'ackdat' of the TCP.
34	synack	float	The time between the SYN and the SYN_ACK packets of the TCP.
35	ackdat	float	The time between the SYN_ACK and the ACK packets of the TCP.

TABLE VI: Time Features

	Feature	**Type**	**Description**
36	is_sm_ips_ports	binary	If source (1) equals to destination (3)IP addresses and port numbers (2)(4) are equal, this variable takes value 1 else 0
37	ct_state_ttl	integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
38	ct_flw_http_mthd	integer	No. of flows that has methods such as Get and Post in http service.
39	is_ftp_login	binary	If the ftp session is accessed by user and password then 1 else 0.
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.

TABLE VII: General Purpose Features

	Feature	Type	Description
41	ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42	ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct_src_ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
47	ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).

TABLE VIII: Connection Features

	Feature	Type	Description
48	attack_cat	nominal	The name of each attack category (e.g., Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms)
49	Label	binary	0 for normal and 1 for attack records

TABLE IX: Labelled Features