UNIVERSITY OF WATERLOO
Faculty of Mathematics

Machine Learning-based DDoS Detection

David R. Cheriton School of Computer Science
Waterloo, Ontario, Canada

Prepared by
Lechuan Peng
4A Computer Science
December 2019

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

This report provides thorough investigation towards network DDoS attack detection. Method of analysis include visualization techniques and machine learning approaches. Results of analysis show that proper machine learning models should be chosen to improve the performance as well as the appropriate hyperparameters. For example, increasing the dropout percentage of the Autoencoder will dramatically increase the metrics for ill-performed attack types, from almost zero to 60%.

1.0 Introduction

Nowadays, the mobile telecommunications is moving swiftly towards 5G networks with more reliable services. E-RAN (Elastic Radio Access Networks) [1] provides flexible and elastic 5G (5^{th} generation mobile networks) network slicing to meet the users' demands by providing different type of services. However, an attack, such as distributed denial-of-service (DDoS), would possibly consume the services of legitimate end-user. Therefore, it is important to secure E-RANs and their services.

The project aims to provide two innovations to secure the E-RAN: *automated intrusion detection* and *automated attack response and mitigation*. Due to the limitation of the time, we only look into the intrusion detection aspect in this report.

However, since the services of E-RAN are not established yet and it is extremely hard to get data from other 5G carrier due to many restrictions, here we propose our solution against CICDDoS2019 dataset [Sharafaldin et al., 2019] from UNB (University of New Brunswick). This dataset remedies the shortcomings and limitations on some existing datasets, with DDoS attacks traces, which is similar to our goal.

This report examines several novel machine learning techniques that can be used for intrusion detection. We use the CICDDoS2019 dataset for our analysis. The remaining part of this report is organized as follows: Section 2.1 introduces Data Visualization, Section 2.2 presents

[1] Ericsson's implementation of C-RAN (Cloud Radio Access Networks)

machine learning models used for anomaly detection, Section 2.3 discusses some machine learning models capable of attack type classification and Section 3.0 is the conclusion.

2.0 Analysis

Before moving to the analysis part, we shall briefly mention the general structure of CICDDoS2019 dataset [Sharafaldin et al., 2019] we were working with as well as methodologies they are using.

The CIC (Canadian Institute for Cybersecurity) team of UNB [1] generate a new dataset, namely CICDDoS2019, which contains several types of DDoS attacks. They first generate raw pcap [2] files, then pass it to CICFlowMeter [3] to generate a set of features for traffic flows, such as the information of forward/backward packets and the number of SYN (Synchronize Sequence Numbers) flag, by generating Bidirection Flows and aggregate associated packets. The results are CSV (Comma-Separated Values) files. The way they label data is following the specific time constraints and IP address (Internet Protocol Address). In their experiment, they label the traffic as benign, which is also known as background traffic, as opposed to attack traffic, if the neither Destination IP nor Source IP is from third party attacker IP. Otherwise, the traffic is marked as attack, where the attack type follow the time period they launch different attack types.

Two sets of data were generated on two different days. One set is used for training, i.e., train the machine learning models, and it is on January 12th, 2018. The other set is used for testing,

[1] We abbreviate it as UNB team/UNB paper onward.

[2] packet capture

[3] The details of the analyzer and its underlying principles are outlined in [Draper-Gil et al., 2016] and [Lashkari et al., 2017].

Days	Attacks	Number of Flows
Training	NTP	1317073
	DNS	5431885
	LDAP	2592175
	MSSQL	5862391
	NetBIOS	4717785
	SNMP	5632011
	SSDP	2736425
	UDP	3392966
	UDP-Lag	370605
	SYN	1582681
	TFTP	22463459
Testing	LDAP	1915122
	MSSQL	5787453
	NetBIOS	3657497
	PortMap	186960
	SYN	4891500
	UDP	3867155
	UDP-Lag	1873

Table 2.1 Daily Label of Dataset

i.e., evaluate the performance of machine learning models, and it is on March 11th, 2018. In Table 2.1[4], the attacks types and the number of traffics [5] are listed. See Appendix for the full description of the attack types. Also, one can observe from Table 2.1, PortMap, available on test day, has not been seen on training day.

To begin with, we first visualize the data.

2.1 Data Visualization

Data visualization is technique of producing the graphic representation of data. It makes the complex data easier to understand and detect patterns and relationships among different features. We first start with several common techniques and introduce a technique: Topological Data Analysis.

2.1.1 Baseline techniques

When it comes to data visualization, the curse of dimensionality is always the problem since the dimensions that is readable for human are below 3. There are also some common dimensionality reduction techniques: Multidimensional Scaling (MDS), Isomap, Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE) and so on. Here we just experiment the last two, which are commonly used in this field, for the sake of time.

[4]Note the discrepancy from the UNB paper. Here we regard WebDDoS as UDP-Lag attack since UNB paper did not provide enough information on how to extract WebDDoS from CSV file associated with UDP-Lag.

[5]Each attack CSV file contains some benign traffic as well.

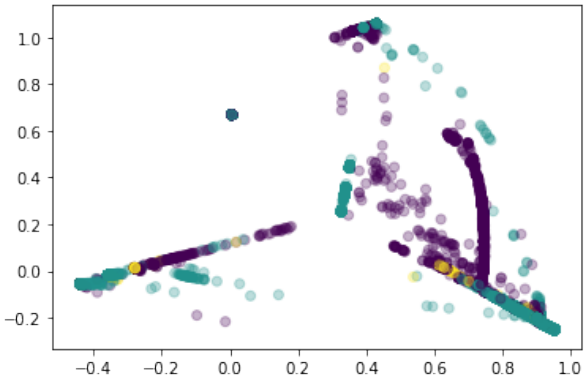

Fig 2.1 Visualize UDP-Lag with PCA

One of the widely-used techniques for dimensionality reduction is Principal Component Analysis (PCA). With PCA, a set of high dimensional data can be linearly transformed into a smaller set of uncorrelated variables that captures most of the information of the original data [Dunteman, 1989]. When dealing with the high dimensionality, most researchers use PCA to perform dimension reduction. Here we perform PCA directly on UDP-Lag [6] on training day. This set of data contains almost 90% UDP-Lag, 7% benign and 3% WebDDoS [7]. In the PCA pipeline, the original high dimension feature space is reduced to 2, with a pair of (x, y) coordinates, then the result is a plot of all these points in a 2D plane. Result is shown in Fig 2.1.

Not too surprisingly, we cannot see a good separation between these traffics, i.e., different type of points are mixed across the whole area. This is because the features of traffic flows are not linearly correlated and lots of information are lost during the projection procedure of PCA. Then we would like to discover some non-linear relationships between different features.

In [Maaten and Hinton, 2008], authors proposed a new non-linear dimension reduction technique: t-Distributed Stochastic Neighbor Embedding (t-SNE). This is a relatively novel technique because it uses the idea from *Student t-distribution* to reconstruct the probability dis-

[6] Due to the memory issue, we did not do experiments on all attack types.

[7] We adopt the color notation as follows: purple for UDP-Lag, green for benign, yellow for WebDDoS.

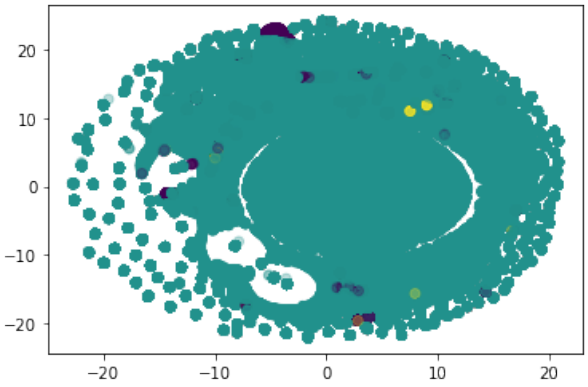

Fig 2.2 Visualize UDP-Lag with t-SNE

tribution in low-dimensional space. Moreover, it leverages the *gradient descent* to optimize the embedding, reducing the risk of getting stuck in local minima. In comparison with PCA, many researchers choose t-SNE over PCA because it captures some non-linear relationships and work well in practice. Visualization is shown in Fig 2.2.

From Fig 2.2, we can see that UDP-Lag traffics are distributed all over the plane, and has a large percentage of overlap with benign and WebDDoS. Therefore, t-SNE is not suitable for this dataset either.

As we can see from the results, traditional methods fail to exploit the correlations between different features. Then we can move our steps to TDA.

2.1.2 TDA

Topological Data Analysis (TDA) is a novel technique to visualize the data. It aims to provide well-founded mathematical, statistical and algorithmic methods to infer, analyze the hidden topological and geometric structure underlying high dimensional data [Chazal and Michel, 2017].

In the field of TDA, there are a number of branches which study different aspects of data, as well as python libraries [Saul and Tralie, 2019]. In this report, we introduce one TDA pipeline originating from persistent homology [8].

Here we introduce the basic pipeline from [Lum et al., 2013]:

1. Apply two filter functions [9] to the original data and get a set of points in 2D plane.
    - Note that the first filter function should highlight special features and second filter should disperse the data as opposed to the first filter function [van Veen and Saul, 2019].
2. Align a set of hyperboxes (here we use squares) to cover all points with an overlap percentage p.
3. Apply a clustering scheme to each square.
4. Label each cluster as a node. The size and the color of the node is determined by the number of points and distribution of the points (malicious or benign traffic), respectively.
5. Connect the nodes with an edge if they share points in common.

Let us walk through some steps of the pipeline. In the first step, a set of points in 2D plane is obtained. Secondly, we align all boxes $\{B_i\}_{i\in I}$ to cover all points $\{x_j\}_{j\in J}$ in the plane. Mathematically, it can be represented as follows:

[8] a branch of TDA, which studies the topological features, such as persistence, under different spatial resolutions or scaling parameters [Wikipedia contributors, 2019].

[9] Filter function is defined to map high dimension to single dimension: $f : \mathbf{R}^n \to \mathbf{R}$

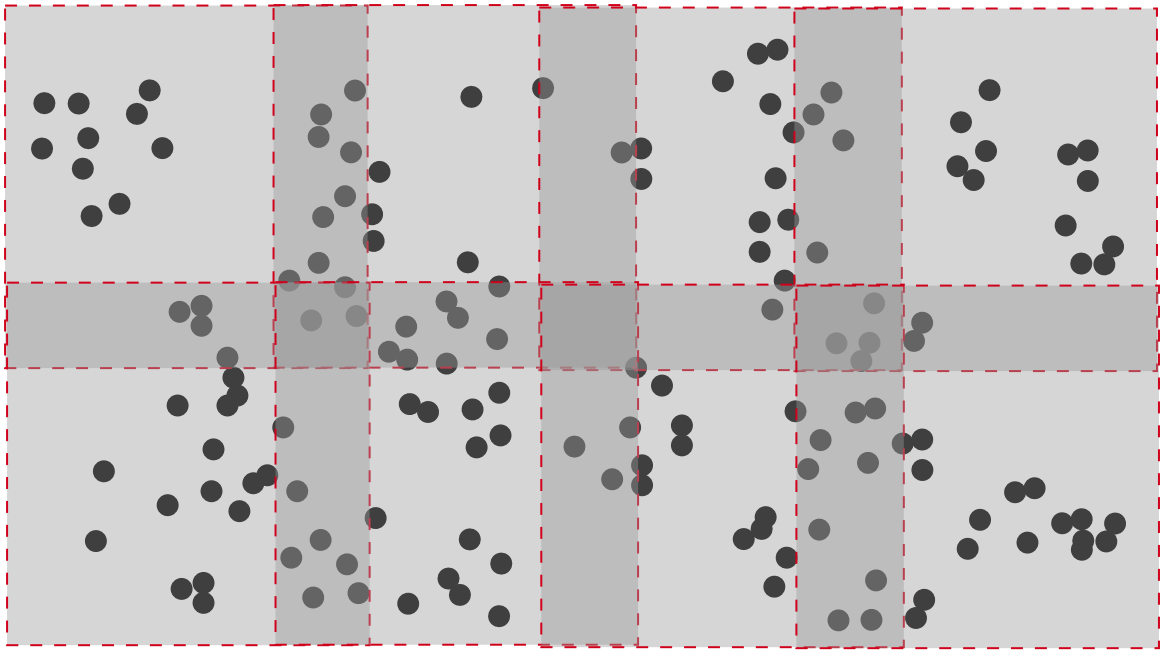

Fig 2.3 Intermediate steps of TDA pipeline

$$\forall j \in J, \;\; x_j \in \bigcup_{i \in I} B_i$$

Also, Fig 2.3 shows how boxes are aligned to cover all the points with a certain percentage of overlap. Then each box is applied with clustering scheme to produce some clusters (represented as colored nodes in this case) inside each box. The last step is to connect the nodes with edges, which can provide more information on the relationship between neighbor nodes.

This is a slight variation, with addition of clustering scheme, of the original mapper algorithm [Singh et al., 2007], which is founded on the nerve theorem:

Theorem (Corollary 4G.3 [Hatcher, 2002]). If U is an open cover of a paracompact space X such that every nonempty intersection of finitely many sets in U is contractible, then X is homotopy equivalent to the nerve $N(X)$.

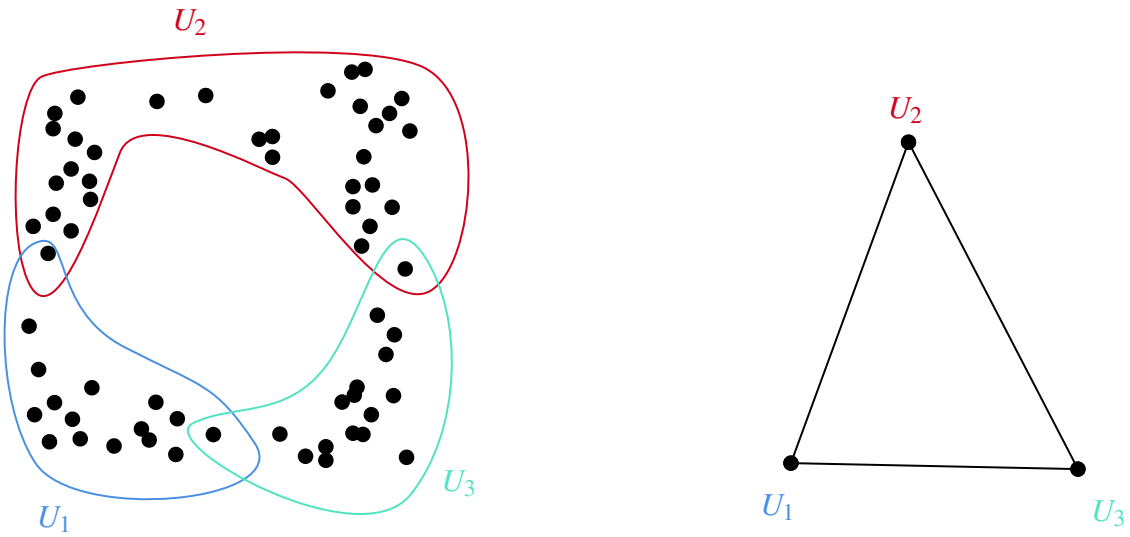

Fig 2.4 Graphic intuition of the nerve theorem

Fig 2.4 provides an intuition of the theorem, the original data (on the left) is homotopy equivalent to the nerve (on the right).

Therefore, the final visualization does capture most of the features of the data.

With the Kepler-Mapper [van Veen and Saul, 2019], we can visualize the UNB dataset by applying the TDA pipeline as shown in Fig 2.5. Here we choose Isolation Forest [Liu et al., 2008] and median as two filter functions and K-Means [Steinley, 2006] as the clustering algorithm. Isolation Forest is a widely used anomaly detection machine learning models which can separate the outliers out of the majority of the dataset. This works well in our experiment since it highlights the special feature (anomaly score) of the input data as stated in TDA pipeline. K-Means a light clustering algorithm in terms of the computational complexity. Due to the limit of machine, we did not experiment with other computational heavy clustering (Density-Based Spatial Clustering of Applications with Noise) and filter algorithms (k-nearest neighbor) on the whole dataset.

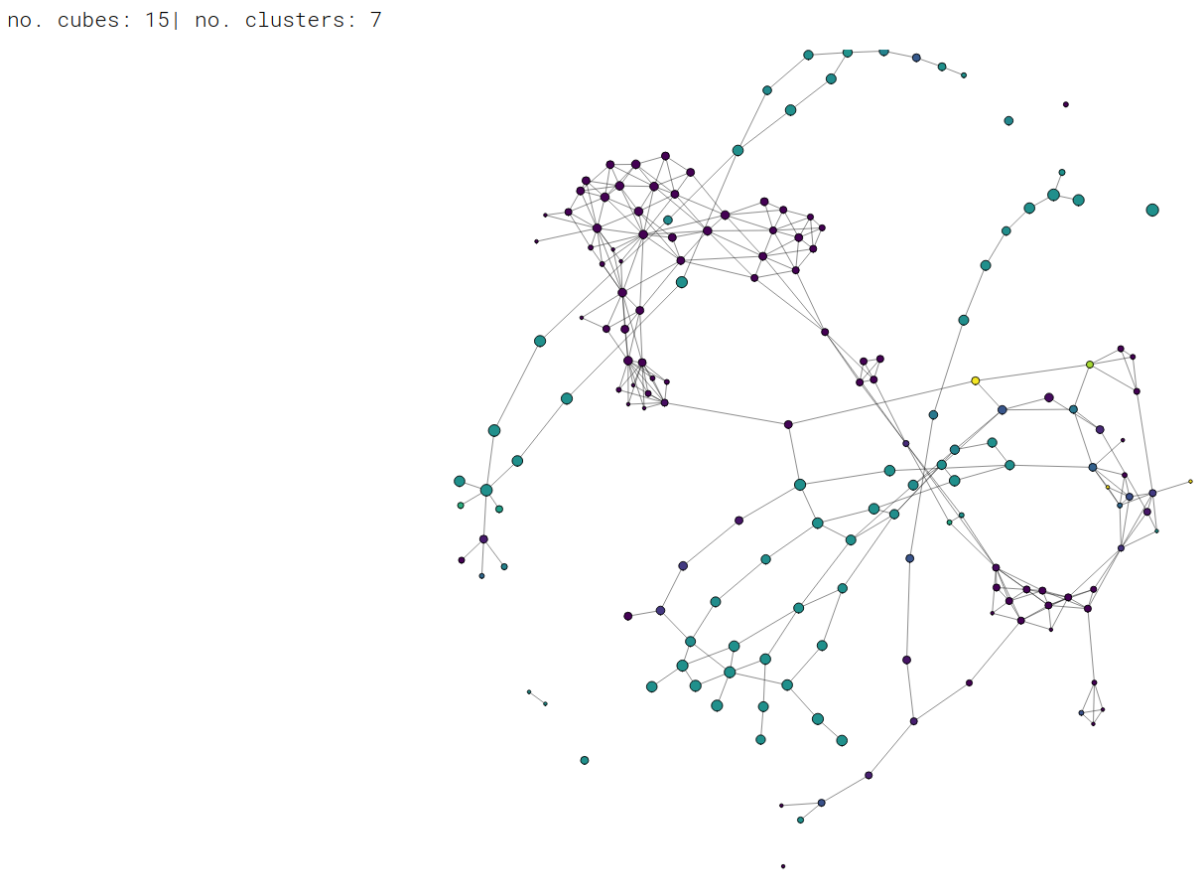

Fig 2.5 Visualize UDP-Lag with TDA

In Fig 2.5, in comparison with Fig 2.1 and Fig 2.2, we can see a mostly complete separation between different type of traffics by their colors. The benign traffics (in purple) are located at the top-left and bottom-right, which are connected by edges. On the other hand, attacks (in green and yellow) are located on the other sections of the graph.

This visualization is supported by D3.js [Bostock et al., 2011]. This JavaScript library disperses the data in dense area with the "gravity" feature, which stands out from other traditional visualization techniques.

Moreover, the performance of this TDA visualization is mostly determined by the choice of the filter functions which project the data to 1D space. Here the choice of Isolation Forest did separate the minority (benign) from the majority (attack). Poor choice of filter functions will result in two consequences: attack and benign cluster nodes are not separated spatially; cluster nodes are not pure, i.e., it is not mainly contributed by one type, either attack or benign.

Here we adopt TDA as a tool to see how our NIDS (network intrusion detection system) performs by seeing the visualization. In the next section, we leverage state-of-the-art machine learning techniques to identify attack traffic from benign traffic as well as some classification tasks. Combined with TDA, the performance of machine learning classification is easily evaluated by the visualization.

2.2 Attack Detection

Over the years, many machine learning approaches have been used intrusion detection. For example, isolation forest, PCA, Logistic Regression and so on. Here we propose a solution using Autoencoder for binary classification, i.e., to distinguish attack from benign.

The main motivation of using Autoencoder is that it can learn the complex patterns or features of one specific type, either linear or non-linear relationships between different features, then Autoencoder will use this set of patterns to distinguish the patterns Autoencoder has not seen yet.

2.2.1 Introduction to Autoencoder

Autoencoder is artificial neural network (ANN) which can learn the coding of the input data in an unsupervised manner [Géron, 2019].

In general, Autoencoder has two parts: encoder and decoder. Encoder tries to “encode" the input data and gets the latent representation[10]; while decoder tries to “decode" the latent representation and recovers the original input data as depicted in Fig 2.6. Here x_i where $i = 1, 2, 3, \ldots$ denotes one specific feature of an input feature vector $\mathbf{x} = [x_1\ x_2\ \cdots]^T$. For example, x_1 denotes the packet length of one traffic flow, x_2 denotes the Flow IAT Mean (Mean time between two packets sent in the flow) and so on. On the other head, x'_i where $i = 1, 2, 3, \ldots$ denotes the **reconstructed** feature of an output feature vector $\mathbf{x}' = [x'_1\ x'_2\ \cdots]^T$. Therefore, during the training

[10] a subspace of input feature space

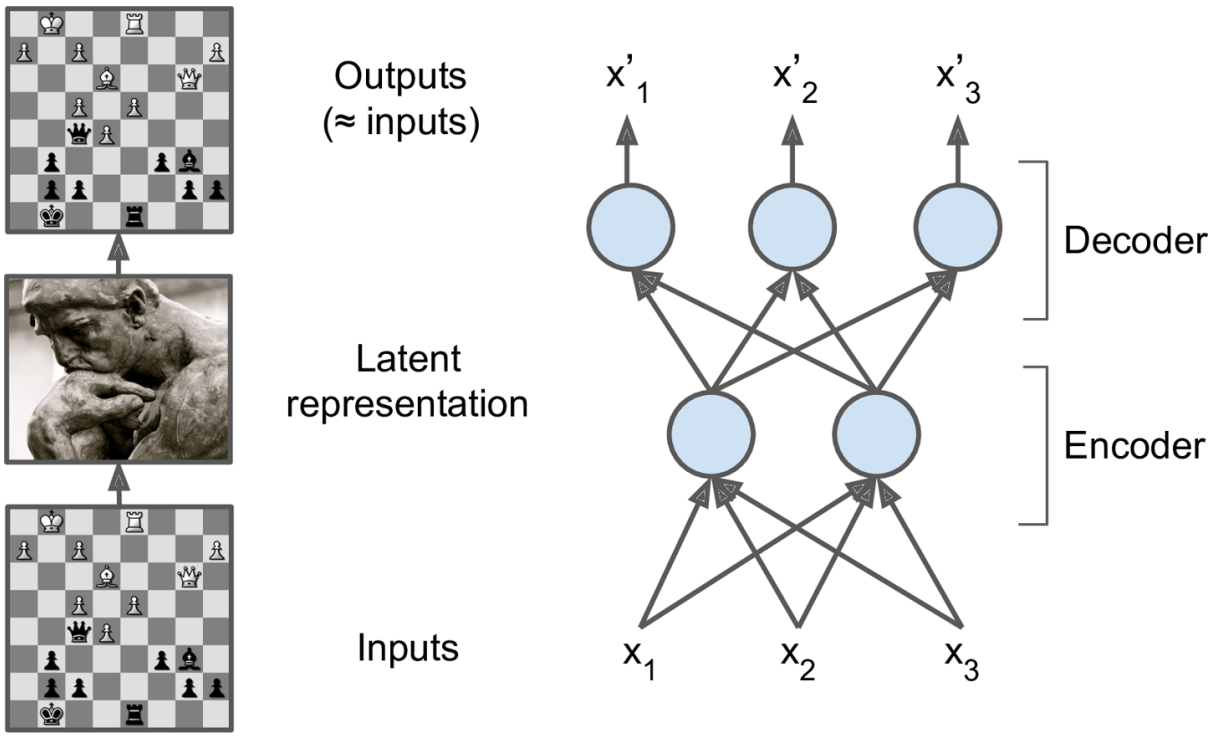

Fig 2.6 The chess memory experiment (left) and a simple Autoencoder (right). Source: [Géron, 2019]

Out inputs are pure benign traffics.

phase, Autoencoder tries to reconstruct the inputs perfectly, or in other words, minimizing the construction error as defined formally later.

The chess memory experiment in Fig 2.6 learns the pattern of the chess arrangements. In our context, i.e. anomaly detection, we let Autoencoder to learn the pattern of benign traffic: successfully encode and decode the input data. Since it has not seen the pattern of the attack traffic before, it will not be able to encode and decode of the attack traffic flow, therefore Autoencoder will mark it as attack.

2.2.2 Autoencoder experiments

As mentioned in previous subsection, Autoencoder only learns the pattern of benign traffic. So in our model, we train our Autoencoder with benign traffic only. During the training process, it learns how to encode and decode the input benign traffic. Then we compare the input and output and get a reconstruction error. Here we adopt *mean square error* (MSE):

$$MSE = \frac{\sum_{i=1}^{n}(x_i - x_i')^2}{n}$$

where $\mathbf{x} \in \mathbf{R}^n$ denotes the input feature vector and $\mathbf{x}' \in \mathbf{R}^n$ denotes the output/reconstruction vector.

The big picture of one Autoencoder is organized as follows:

- Choose an Autoencoder (AE) with necessary parameters: structure, activation function and so on (details later).
- Split the benign traffic into 2 sets: training (trn) set and optimization (opt) set.
- Train AE with the training set, given input parameters such as the learning rate (details later).
- Use optimization set to produce a *threshold* (tr): distinguish attack traffics from benign traffics. The formula is adopted from [Meidan et al., 2018]:

$$tr^* = \overline{MSE}_{DS_{opt}} + sdv\left(MSE_{DS_{opt}}\right)$$

AE structure [11]	bios, ownae, deeper
learning rate	0.0001 to 0.1
number of epochs	100, 150, 200
optimizer	sgd, adam, adamax, rmsprop, adagrad, adadelta, nadam
activation function	ReLU, tanh
dropouts	no dropout, 10%, 20%, 30%, 50%

Table 2.2 All possible combinations of AE models

where *sdv* denotes the *standard deviation* and DS_{opt} is the dataset split for optimization. If the MSE is above this anomaly threshold, then AE predicts this flow as attack, otherwise benign.

Now we can evaluate the performances of different models with all possible combinations on hyperparameters as well as other structures as in Table 2.2. Here learning rate is the step size between each learning epoch. Epoch is a complete pass through the entire dataset. Optimizer is a hyperparameter specifying how AE learns, how to find the optimal. The activation function of a neuron defines the output of that neuron with the input data. Dropout is a common techinque to avoid overfitting the data by randomly droping out the neurons in the hidden layer.

In our experiments, we tried different combinations as in Table 2.2. Here the implementation is based on Keras [Chollet et al., 2015]. At this step, we evaluated our performance on the attacks of training day as a validation process. It turned out that the Autoencoder structure from [Meidan et al., 2018] (*bios* for short-hand notation) together with *Adamax* [Kingma and Ba, 2014] optimizer had the best performance (with other necessary arguments fixed) [12]. We evaluated

[12] See Appendices for full experiments setup. Here we leave out the details of comparison between all combination due to the information capacity of this report.

several metrics specifically: *F1-score, precision, recall*. See Appendices for details. Here the most important metric is recall[13]. Here, in our case, the lower the recall, the more attacks are mislabeled as benign. Our NIDS, as the first defense, should be able to identify most attacks. The experts will only analyze the traffics marked as benign later, then as a consequence, attack traffics are not correctly detected. Since we are mostly interested in *recall* and other metrics did not show any variations across different models or correlated with *recall*, we shall present *recall* value as the main evaluation metric here.

However, the model currently presented (bios structure and Adamax optimizer) might overfit the validation data. Then we performed dropouts. The comparison results are shown in Fig 2.7. Here the results are average performance across repetitive experiments due to the random initialization of TensorFlow backend [Abadi et al., 2015] and this idea is used for the following experiments as well.

From the results, we can observe that decent percentage (10%, 20%) dropouts can improve some of the ill-performed attack types, (MSSQL, NTP, SSDP and UDP) with a significant amount of recall values. However, in general, Autoencoder did not perform well on attacks like NetBIOS even with dropouts. The reason might be the insufficient amount of trained benign traffic. Compared with attack traffic, it is around millions times less. Also, the pattern of these attack traffics might be similar to benign traffics.

The next step is to compare evaluate our model against the traffic on testing day and compare our results with the results in [Sharafaldin et al., 2019] paper.

Within testing phase, we first ran the model against each attack type on test day. The recall

[13]Fraction of attacks that were correctly identified

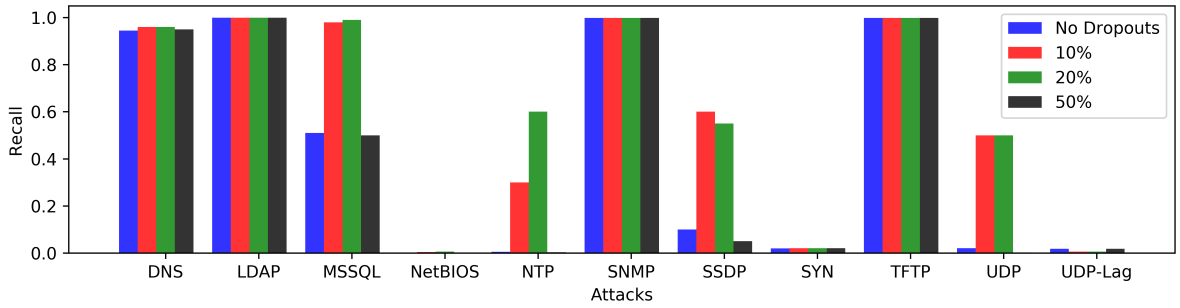

Fig 2.7 Recall values for different models against different attack types (validation set on training day)

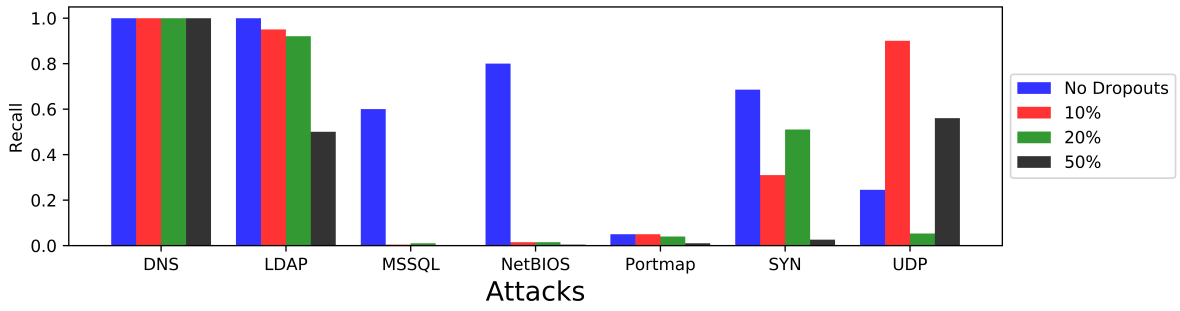

Fig 2.8 Recall values for different models against different attack types (the complete set on testing day)

values are depicted in Fig 2.8. The reason might be that attack patterns on testing day are not overfitted by the Autoencoder. In comparison with Fig 2.7, we can see dropouts have opposite effects on training day and testing day. Therefore, we choose the one without dropouts against testing day traffics. The reason here might be that the patterns for a specific attack, say NetBIOS or MSSQL, are different on training day and testing day.

For the complete testing, we conducted a number of experiments repeatedly to average the performance. The recall values across repeat experiments are shown in Fig 2.9. From this frequency plot, we can see the recall value of AE mostly lies around 0.67 except some outliers. Besides that, precision values are always 0.999.

In comparison with the results in UNB paper [Sharafaldin et al., 2019], as shown in Table 2.3

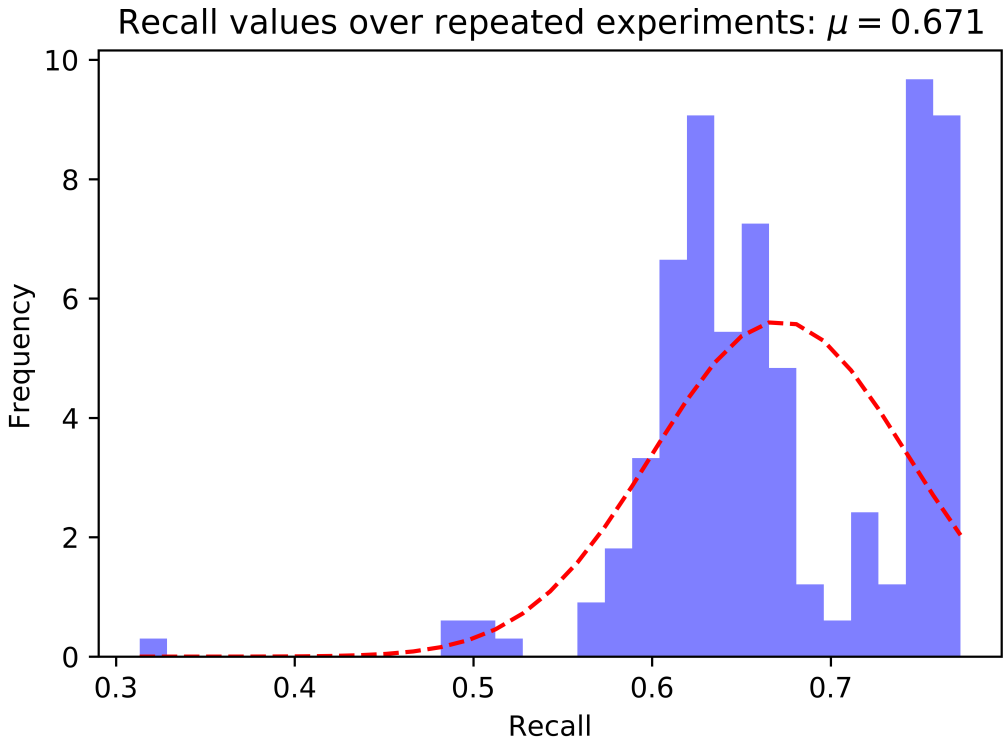

Fig 2.9 Recall values for bios-Adamax model against all traffics on testing day

where they used classic ML models (Random Forest, Logistic Regression, Naive Bayes and ID3), our experiment improves a bit on the recall value, which is what we are mostly interested in.

Alg	Pr	Rc	F1
ID3	0.78	0.65	0.69
RF	0.77	0.56	0.62
Naive Bayes	0.41	0.11	0.05
Log Reg	0.25	0.02	0.04

Table 2.3 The performance examinination results from UNB paper

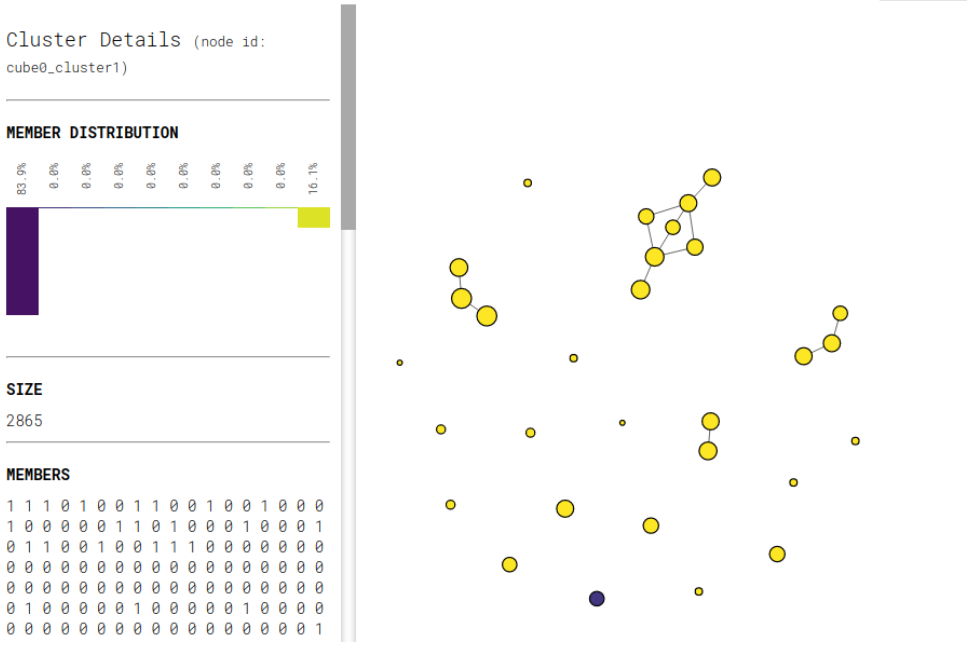

Fig 2.10 Visualizing Autoencoder output with TDA

The left column highlights the composition of the purple cluster at the bottom.

2.2.3 Performance evaluation with TDA

As we mentioned earlier, we can leverage TDA to see how good data are separated (attacks and benign). Previously, we were using isolation forest as anomaly detection tool. In the area of TDA, we used isolation forest as a filter function, together with the other filter, to produce the coordinates. Here we can apply this technique to the output of Autoencoder as well. Recall that the output of Autoencoder is a MSE between original data and reconstructed data. The lower the MSE, the better Autoencoder learns, thus more likely this is a benign traffic, and vice versa.

In our experiments, we feed the output of Autoencoder to the TDA pipeline, together with L^2 norm as another filter function, then we get the final visualization of the data. Fig 2.10 is an example of visualizing the output of Autoencoder with TDA. In this example, we choose to visualize one of the well performed attack types – LDAP, to see how TDA can separate the data. The yellow denotes the LDAP traffic, while purple denotes benign. The left column highlights

the composition of the purple cluster at the bottom, where this cluster mostly contains benign traffic flows, separating from other yellow nodes. Therefore, this visualization does depict how well Autoencoder separates the data, i.e., distinguish attack from benign. This proves Autoencoder is a right choice within the context of detecting LDAP.

2.3 Classification of Attack Types

After we correctly identify one traffic flow as attack, we are also interested the specific attack type of this flow. There are a number of Machine Learning models capable of this task. Here we examine some of these models as well as some optimization techniques.

Unlike attack detection aspects with unsupervised learning, here all models below are in supervised manner, i.e., models are trained with labels.

Here we propose the general pipeline of the (multi-label) attack type classification:

- Split out a set of training data which contains only attacks with labels. Here we filter out only attack traffics on training day to train our classification model.
- Test on the set of testing data and evaluate the performance, which are also pure attack traffics on testing day.

To begin with, we shall first introduce some widely used models, then compare the performance of each model.

Random Forest

Random Forest (RF) [Breiman, 2001] is a model which has an ensemble of decision tree. It then forms a voting system to decide which class should one belong to. The advantage against traditional decision tree is that RF can minimize the error as the number of trees increases.

XGBoost

XGBoost [Chen and Guestrin, 2016] is a Gradient Boosting library which optimizes the existing approaches. Here we only use the classification class in this library. XGBoost has built-in regularizations which avoids overfitting. However, since there are more parameters to train in comparison with RF, training the XGB classifier model takes much longer than RF.

MLP

Multi-Layer Perceptron (MLP) can also perform a multi-class classification task. Here we have one output neuron for each class. With the *softmax* activation function for the whole output layer, each output neuron will output a probability of belonging to this class and the summation of all probabilities are 1. See Fig 2.11 for the basic architecture. [14]

Similar to Autoencoder, MLP can also learn complex patterns due to the complexity of neurons. That is why some researchers choose MLP to classify classes.

[14] See Appendices for full experiments setup

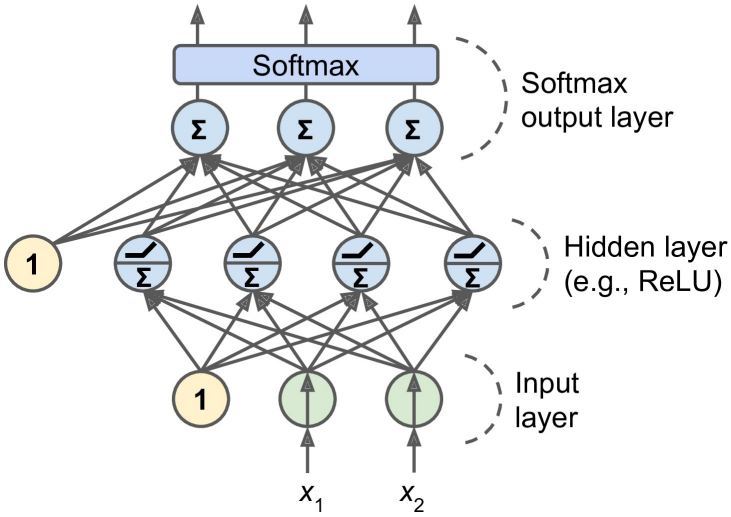

Fig 2.11 A modern MLP for classification. Source: [Géron, 2019]

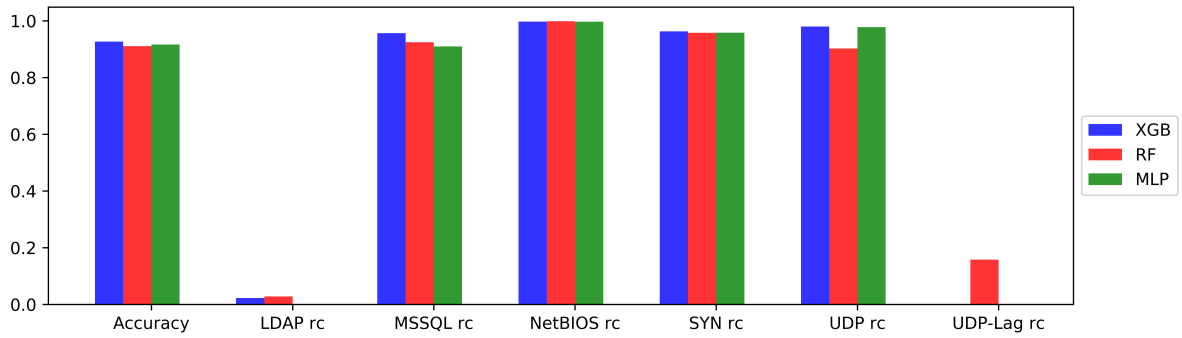

Fig 2.12 Performance evaluation on 3 classification models

2.3.1 Performance

Within this experiment, we present our results in two ways:

- First from the results in Fig 2.12, we can see that all models have similar performances, and they are able to classify most of attack types [15], except for LDAP and UDP-Lag. In Table 2.4, we present the accuracy for three models, where the accuracy indicates the model learns the pattern well.

[15]Here we did not evaluate the performance for Portmap on testing day since this type has not been seen during training process.

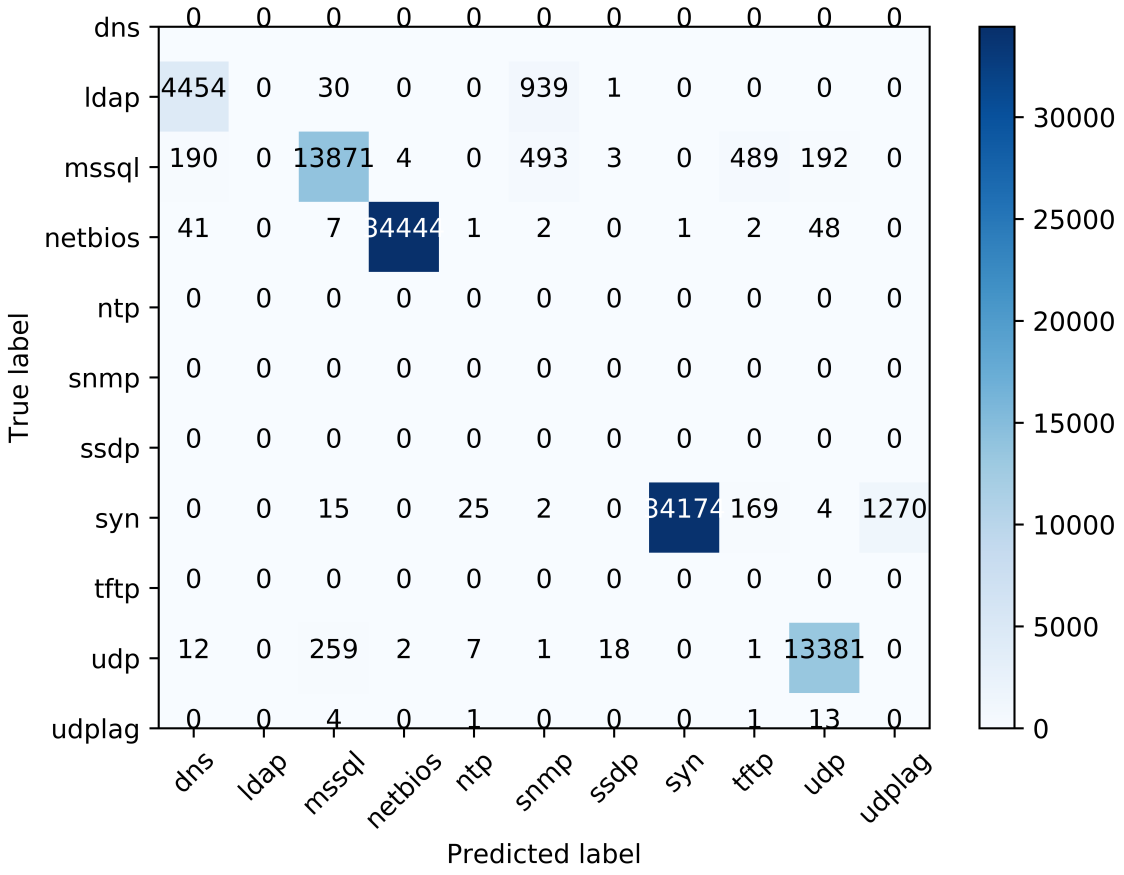

Fig 2.13 confusion matrix for MLP

Classification model	Accuracy
XGB	0.9268
RF	0.9110
MLP	0.9168

Table 2.4 Accuracy for 3 classification models

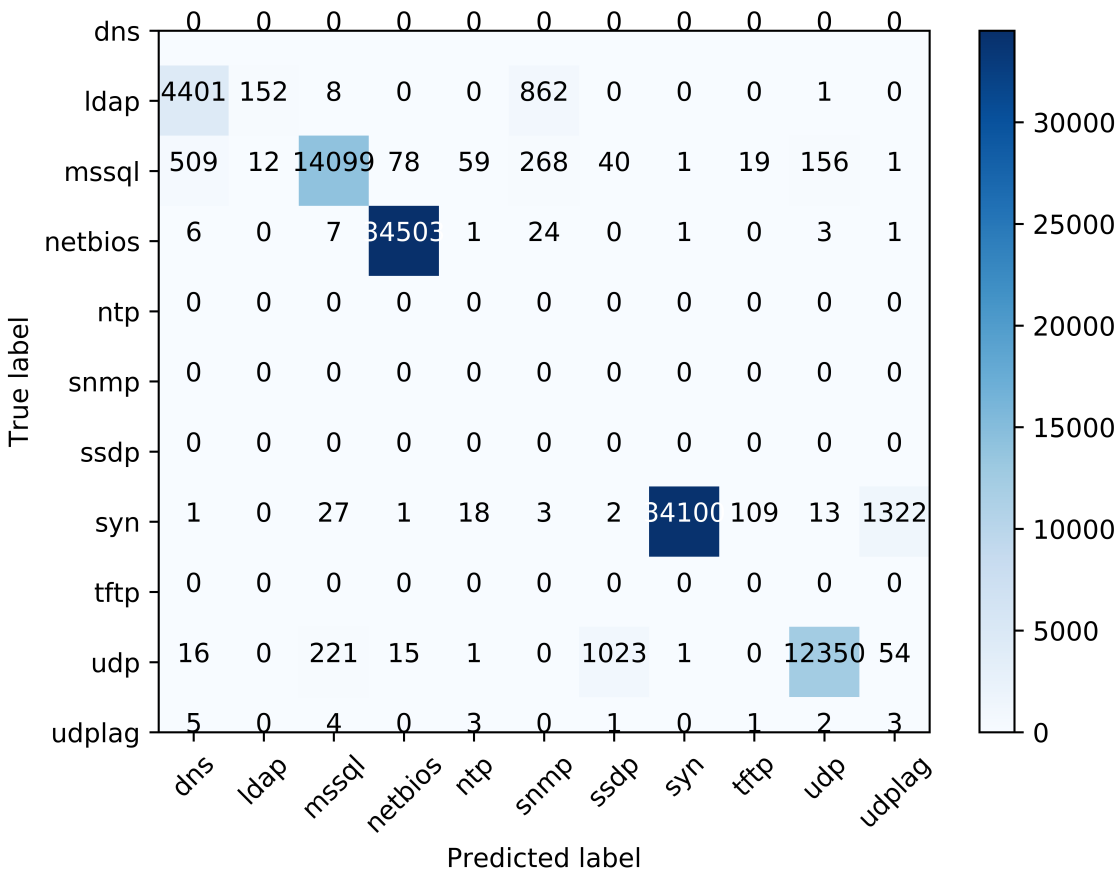

Fig 2.14 confusion matrix for RF

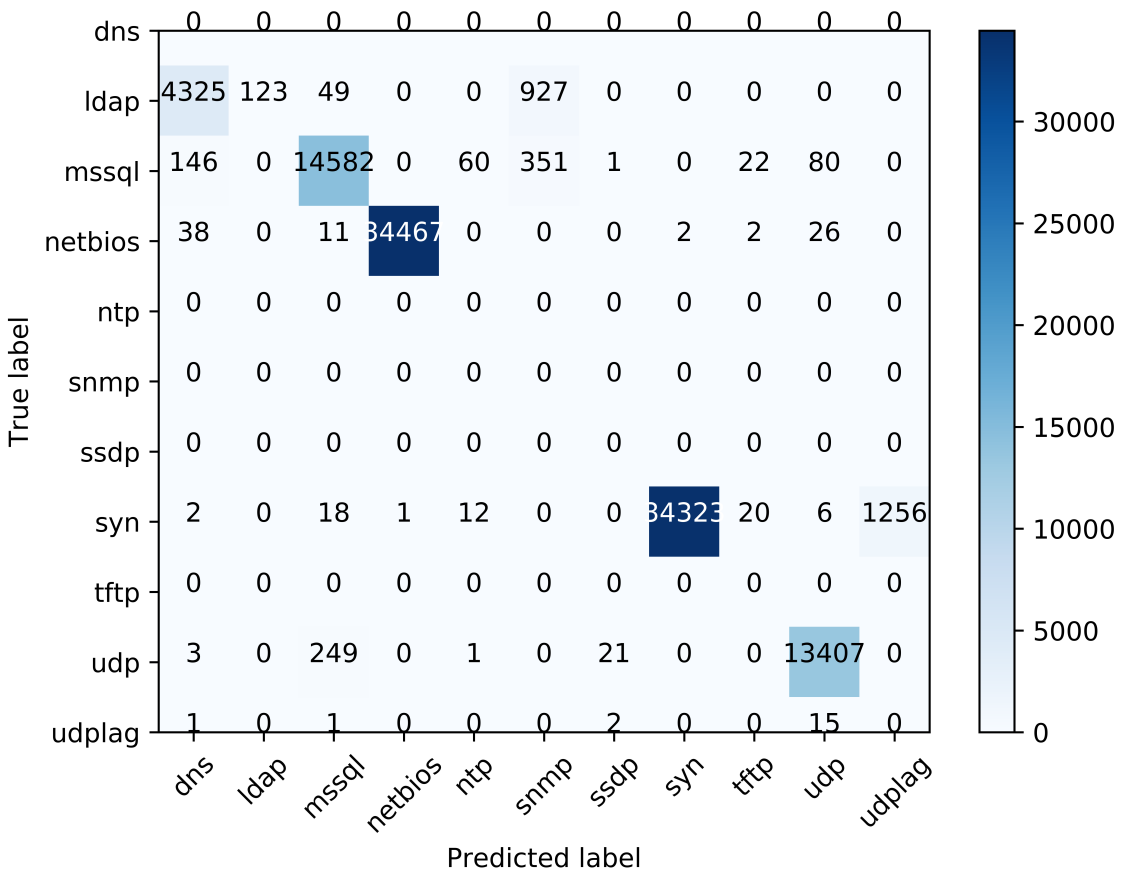

Fig 2.15 confusion matrix for XGB

- Secondly, from the confusion matrix results in Fig 2.13, Fig 2.15 and Fig 2.14, LDAP traffics are mostly mislabeled as DNS and then SNMP. Moreover, in XGB and MLP models, UDP-Lag is recognized as UDP mostly, which might be the similarity between UDP and UDP-Lag.

Because the number mismatch of attack types on testing day in comparison with the training day, i.e., 12 attack types on training day and 7 on testing day, we cannot evaluate the performance for other attack types that are not presented on testing day. In general, all these models prove to have high accuracy on predicting incoming attack traffic. However, it is interesting to note that from RF to MLP to XGB, the accuracy is increasing while the training time is also increasing.

3.0 Conclusions

In this report, we study several machine learning techniques on data visualization, attack detection and attack type classification. Specifically, we apply these techniques on CICDDoS2019 dataset and compare our results with the ones in UNB paper. In terms of attack detection aspects, our presented model, Autoencoder, shows a better result against the classic MLs they are using. After that, TDA shows a good separation if AE's detection works well. Also, our attack type classification can classify the majority of attacks with high accuracy.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Bostock et al., 2011] Bostock, M., Ogievetsky, V., and Heer, J. (2011). D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Chazal and Michel, 2017] Chazal, F. and Michel, B. (2017). An introduction to topological data analysis: fundamental and practical aspects for data scientists. *arXiv preprint arXiv:1710.04019*.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. https://keras.io.

[Draper-Gil et al., 2016] Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I., and Ghorbani, A. A. (2016). Characterization of encrypted and vpn traffic using time-related. In *Pro-*

ceedings of the 2nd international conference on information systems security and privacy (ICISSP), pages 407–414.

[Dunteman, 1989] Dunteman, G. H. (1989). *Principal components analysis*. Number 69. Sage.

[Géron, 2019] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

[Hatcher, 2002] Hatcher, A. (2002). Algebraic topology, cambridge univ. *Press, Cambridge xii*.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Lashkari et al., 2017] Lashkari, A. H., Draper-Gil, G., Mamun, M. S. I., and Ghorbani, A. A. (2017). Characterization of tor traffic using time based features. In *ICISSP*, pages 253–262.

[Liu et al., 2008] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

[Lum et al., 2013] Lum, P. Y., Singh, G., Lehman, A., Ishkanov, T., Vejdemo-Johansson, M., Alagappan, M., Carlsson, J., and Carlsson, G. (2013). Extracting insights from the shape of complex data using topology. *Scientific reports*, 3:1236.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Meidan et al., 2018] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. (2018). N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22.

[Saul and Tralie, 2019] Saul, N. and Tralie, C. (2019). Scikit-tda: Topological data analysis for python.

[Sharafaldin et al., 2019] Sharafaldin, I., Lashkari, A. H., Hakak, S., and Ghorbani, A. A. (2019). Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8.

[Singh et al., 2007] Singh, G., Mémoli, F., and Carlsson, G. E. (2007). Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *SPBG*, pages 91–100.

[Steinley, 2006] Steinley, D. (2006). K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34.

[van Veen and Saul, 2019] van Veen, H. J. and Saul, N. (2019). Keplermapper. http://doi.org/10.5281/zenodo.1054444.

[Wikipedia contributors, 2019] Wikipedia contributors (2019). Persistent homology — Wikipedia, the free encyclopedia. [Online; accessed 20-December-2019].

Acknowledgements

Thanks to the members of Networks Lab for insightful discussions and inspiring friendships.

I truly appreciate to Dr. Raouf Boutaba for his patient mentorship. He has been the best supervisor one can imagine.

I am thankful to Dr. Md Faizul Bari and Dr. Mohammad Ali Salahuddin for their invaluable and detailed guidance throughout this research.

I would like to express my deep gratitude to Dr. Vahid Pourahmadi and Dr. Hyame Alameddine for giving insightful suggestions to my work.

I am extremely grateful for The Cheriton School of Computer Science and Ericsson for giving me the opportunity to do research on state-of-the-art project.

Last but not least, I would like to thank all my friends and my parents for their encouragement and support.

Appendices

Attack Types

Here we introduce selected attack types from CICDDoS2019.

DNS can be carried out using either TCP or UDP.

SYN flooding attacks consumes server resources by exploiting TCP-three-way handshake.

UDP-Lag disrupts the connection between the client and the server.

Performance Evaluation Metrics

- **Precision**: Accuracy of positive predictions
  - Precision = TP/(TP + FP) [1]
  - What percent of your predictions were correct?
- Recall: Fraction of positives that were correctly identified
  - Recall = TP/(TP+FN)
  - What percent of the positive cases did you catch?

[1] T/F: True, False. P/N: Postive, Negative.

- F_1-score: The harmonic mean of precision and recall

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Experiments Setup

Computer configuration

Name:	ASUS SABERTOOTH 990FX R2.0
CPU:	AMD X8 FX-8320 3.80GHZ (8 Core)
RAM:	16GB
HDD:	2TB

Note For the two MLP below, we fix the size of input feature to be 77, where we filter out some bias information: Destination/Source IP/Port, Timestamp and some invalid columns.

Autoencoder

Bios structure in [Meidan et al., 2018]:

- Encoder: 75%, 50%, 33% and 25% of the input layer's dimension.
- Decoder: 33%, 50%, 75% of the input layer's dimension.

and together with ReLU as activation function, Adamax as optimizer, default learning rate (0.002) for Adamax, no dropouts, 150 epochs and EarlyStopping in Keras callbacks, achieved the best result among all possible combinations.

Other AE structures

- **ownae**
  - Encoder: 80%, 70%, 60% and 50% of the input layer's dimension.
  - Decoder: 60%, 70%, 80% of the input layer's dimension.
- **deeper**. Denote the input layer's dimension by n.
  - Encoder: $n, n-1, n-2, \ldots, \lceil \frac{n}{2} \rceil$
  - Decoder: $\lceil \frac{n}{2} \rceil + 1, \lceil \frac{n}{2} \rceil + 2, \ldots, n-1$

Classification MLP

For the all hidden layers, we use ReLU as an activation function. For the output layer, *softmax* is used as an activation function. The architecture is as follows:

$$77 \to 100 \to 150 \to 200 \to 12.$$

Due to the memory limit of local machine, we did not perform testing on whole data. Instead, we perform some stratified sampling to avoid bias.