DGA Detection with Machine Learning and Deep Learning

Lixi Zhou, Haimin Zhang, Shiqi Dai
{zhou.lix, zhang.haim, dai.shi}@husky.neu.edu
CSYE 7245, Fall 2018, Northeastern University

1. Abstract

DGAs can constantly generate large amounts of domains to evade blacklist detection. It is hard to block the connection between hackers and zombie computers in real-time with traditional ways. In order to solve this problem, we decided to use machine learning algorithms to detect DGAs and compare the performance of these algorithms. In this research project, we first performed feature engineering. Then applied preprocessed data to machine learning models like a random forest, support vector machine, naive Bayes classifier. Meanwhile we also tried CNN and LSTM. The goal of this project is to acquire model(s) which achieve the classification accuracy higher than 98%. The result is our LSTM model achieved 98.56% accuracy and H2O ensemble achieved 97.34% accuracy.

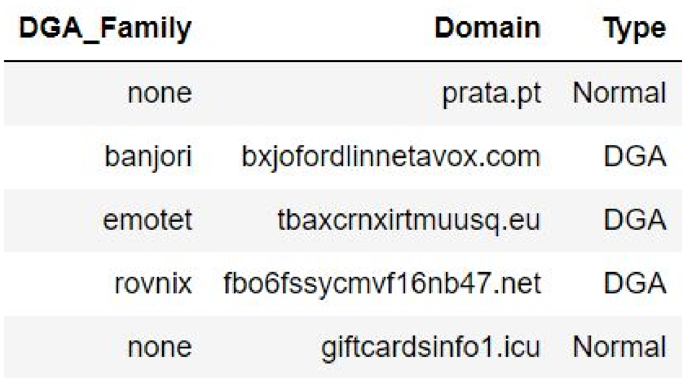
2. Introduction

Internet security vendors have provided several strategies to intercept DGA traffic. In traditional, security providers would first decode the algorithm by applying reverse engineering. Generating a list of domains with potential C2 traffic. Another common strategy is to find similar domain groups by using their statistical properties to determine if DGA generates a domain. The main disadvantage of traditional strategies is the lack of capability to be used for real-time detection and protection.

Therefore, several strategies based on machine learning are introduced. FANCI is one of them. FANCI stands for Feature-based Automated NXDomain Classification and Intelligence, and it was introduced in 2018 by Schüppen, Teubert, Herrmann, and Meyer. It is a system for detecting infections with domain generation algorithm based malware by monitoring non-existent domain responses. FANCI mainly uses two supervised learning algorithms, random forests and support vector machine. Because of the using of RF and SVM, all the domain data(text) has to be represented by features. Schüppen et al.,(2018) described 21 features and used three different categories to group their features. They are structural features, linguistic features, and statistical features. Structural features have to subcategories as inherent structural features and non-self-explanatory structural features. Non-self-explanatory structural features can be some boolean type features or calculated ratio features. Linguistic features are used to measure the deviations from common linguistic patterns of domain names. Statistical features are n-gram frequency distribution and entropy which are common approaches in the feature engineering of domain data.

However, the above traditional machine learning approaches have to use manually picked features to create classifiers like FANCI. They usually have two significant drawbacks: First, hand-crafted features are easy to circumvent by hackers. Second, getting hand-crafted features is relatively time-consuming at the runtime. Thus deep learning/neural network approaches have been taken seriously nowadays. Specific neural networks require less feature engineering and perform better at the run-time. They work directly on raw domain names with a minimal transformation. In other words, if a new family of DGA shows up, the classifier can be retrained right away without the need for manual feature engineering. Also, neural network models act like the black box so it is hard for hackers to reverse and beat. Second, deep learning models have better "true positive"/"false positive" rate and real-time performance. In test cases, neural network classifiers are usually able to achieve satisfying accuracy.

For this research, we have three datasets in total: one dataset for Benign Domains, Alexa Top 1 Million Sites, which are a combination of good domains; two datasets for DGA Domains (https://www.kaggle.com/cheedcheed/top1m), Bambenek Consulting provided malicious algorithmically-generated domains (http://osint.bambenekconsulting.com/feeds/dga-feed.txt) and 360 Lab DGA Domains (https://data.netlab.360.com/feeds/dga/dga.txt). Combing these three datasets and shuffling them to generate the dataset for the project.



DGA_Family	Domain	Type
none	prata.pt	Normal
banjori	bxjofordlinnetavox.com	DGA
emotet	tbaxcrnxirtmuusq.eu	DGA
rovnix	fbo6fssycmvf16nb47.net	DGA
none	giftcardsinfo1.icu	Normal

Figure 1. Original dataset

3. Methods

3.1 Data Representation

Our data are domains which can be seen as semi-structured data. It is not like structured data which has fixed number and type of variables, neither like unstructured data from which we can barely find rules. In the real world, structured data is usually presented as tables such as most of financial data. Every detail of structured data is straight-forward. We can get the data structure and information quickly. Structured data is logical, inductive. It is rich of human intelligence. On the other hand, unstructured data is often about observed nature things, such as images, sounds, languages, etc. People used typically numbers to represent those data. Thus, we can hardly get any knowledge from the data(values) itself, unless decoding it with some tools, for instance, computers. Therefore, unstructured data is intuitive, deductive. It is about the ways people interpret the data, represent the data by a uniform method.

Semi-structured data is a special one right between those two above. It usually yields to some rules which makes it logical for people to read. However, semi-structured data has it flexibility once obeying to the rules. This means, even we can easily find some rules in a group of semi-structured data, we can’t actually get all the senses of it. Semi-structured data has the property of self-describing. Typical semi-structured data includes mark languages like HTML, XML, system log/trace files, and of course our data—domains. When reading semi-structured data, it is common to have this feeling that just knowing it half well. For example, when we read a raw html file, we can easily find some rule such as tagging, parameters, indention, etc., but we cannot really get the meaning from it. This kind of data is not what you can intuitively interpret. It requires more related knowledge. Thus, semi-structured data is hard for computers processing. Maybe you heard expert system which uses tons of complex rules to let the system “have” knowledge. That is one approach that people was trying to use in dealing with difficult problems like processing semi-structured data. That is beyond our scope of this project.

In our project, we try to use machine learning and deep learning techniques on handling data. So there are two approaches we can use. One, converting semi-structured data to structured data. Two, taking them as unstructured data. Algorithms of traditional machine learning work well with structured data. Most of them has good interpretability on predictions. On the other hand, deep learning algorithms can process unstructured data as the way the brain processing things. These so-called artificial neural networks is able to building high-level logical mapping between the input unstructured data and the final out by extracting features from different layers.

Based on the different properties traditional machine learning and deep learning own, we decided to convert our domain data to a structured table and feed it into traditional machine learning algorithms, meanwhile find a way represent the domains and feed the representation as unstructured data into neural networks. More details are introduced in next parts of this paper.

3.2 Feature Engineering

For the machine learning part, only the attribute of the domain itself is not enough for a machine learning algorithm. It needs some features. Applying features engineering first. Based on our knowledge and reference materials, three kinds of features will be generated: Structural Features; Linguistic Features; Statistical Features. For the first part of feature engineering:

Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
DNL (Domain Name Length)	8	19
NoS (Number of Subdomains)	1	1
SLM (Subdomain Length Mean)	5.0	16.0
HwP (Has www Prefix)	0	0
HVTLD (Has a Valid Top Level Domain)	1	1
CSCS (Contains Single-Character Subdomain)	0	0
CTS (Contains Top Level Domain as Subdomain)	0	0
UR (Underscore Ratio)	0.0	0.0
CIPA (Contains IP Address)	0	0

Table 1. Structural Features

From Table 1, nine structural features are generated. For the example, prata.pt, DNL (The length of the domain name) is 8. It only has 1 subdomain, so its NoS value is 1. The length of the subdomain (SLM) is the length of ‘prata’, which equals to 5.0. It does not have www Prefix, so its Hwp value is 0. ‘.pt’ is a valid top-level domain, so its HVTLD domain is 1. It does not contain single-character subdomain, so the CSCS value is 0. So does the CTS. The ratio of underscore (UR) for example is 0 also. And it does not have an IP address.

Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
contains_digit (Contains digit)	0	0
Vowel_ratio (The ratio of vowel)	0.4	0.25
Digit_ratio (The ratio of vowel)	0.33	0.0

Table 2. Linguistic Features

Based on linguistic analysis, three linguistic features are generated from the domain. Whether a domain contains a digit (contains_digit), the ratio of the vowel in a domain and the ratio of the digit. The value of these linguistic features can be known from Table 2.

Features	Ex: prata.pt	Ex: tbaxcrnxirtmuusq.eu
RRC (The ratio of repeated characters in a subdomain)	0.25	0.33
RCC (The ratio of consecutive consonants)	0.4	0.625
RCD (The ratio of consecutive digits)	0	0
Entropy (The entropy of subdomain)	1.92	3.5

Table 3. Statistical Features

There are also 4 statistical features will be generated. From Table 3. RRC represents the ratio of repeated characters in a subdomain. RCC represents the ratio of consecutive consonants, RCD represents the ratio of consecutive digits and Entropy means the entropy of subdomain.

3.3 AutoML: H2O

H2O has an industry leading AutoML functionality that automatically runs through all the algorithms and their hyperparameters to produce a leaderboard of the best models. One disadvantage of the H2O Auto Machine Learning is that because it needs to use a lot of models to find the best model of the dataset, so it will take a lot of time to find the best model.

3.4 Random Forest

Random forest is a bunch of decision trees. It can be seen as an ensemble model. A random forest model will take all predicting results from its inner decision trees as a vote.

3.5 Naive Bayes Classifier

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Above,

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

Build Naive Bayes Classifiers

Again, scikit learn (python library) will help here to build a Naive Bayes model in Python. There are three types of Naive Bayes model under scikit learn library:

Gaussian: It is used in classification and it assumes that features follow a normal distribution.
Multinomial: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".
Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

3.6 Neural Network Models

3.6.1 Preprocessing

Embedding

Before, we introduced the representation strategies for different types of data and approaches we applied in our project. In this part, we would give more details on the representation of domain data. We’ve mentioned that neural networks are good at processing unstructured data which has already represented in some way. We can name this representation embedding.

Normally we have two methods for embedding. One is to build some index mapping the features to indices. Based on this idea, we can either directly use the indices as sequence or further convert them to one-hot encoding. On the other hand, we can use a n-dimensional vector to encode a single feature. The way to generate this vector is based on different algorithm. Some take the context of features to calculate, others may take some statistics from the document to calculate. In general, we call these methods which using n-D vectors distributed embedding. The main idea of distributed embedding is to use high-level logics to represent data. Typical distributed embedding methods are word2vec, TF-IDF, etc. But they are mainly used in natural language processing.
For our domain data, it is not like natural language which you can easily tokenize. On the contrary, there is no way to tokenize a domain that helps on further processing. Domains do not contain contextual logics. Take our DGA domain as an instance. A DGA domain usually consists by random combination of characters and numbers. There are no relations between digits next to each other. We would never find a tokenization criterion for tokenizing DGA domains. Understanding this, it is easier to know the reason we chose character-level embedding—character to sequence.

To implement character to sequence, we need firstly build a dictionary to store all unique characters (including numbers and signs). The dictionary would be like the index. Once we got it, we can converting our domains character by character to indices corresponding to the dictionary. Note that we should leave index 0 out for later using zero as the padding value.

Padding

Because the input shape of neural networks is required to be uniform, we need to pad all the samples(domains) to a specific length. In other words, padding is, normally, to add zeros to make the input sample vectors have a same length. There are two choices for us to pad:
1. Choose a length that covers most of samples.
2. Choose a length that covers all the samples.

Pros and cons rise accordingly. For the first one, imagine a situation in which the lengths of data is unbalanced distributed. Most of the data have moderate lengths, and only a small part has long lengths. If we plot this distribution out, it will look similar to a lognormal distribution.

In this case, when we choose a length not intending to cover all the lengths, the padding length can be much smaller than choosing the max length which covers all the data lengths. Because this padding length can cover most of our data, we can still represent the data well without losing too much information (only those extra-long data would be cut-off), which means, by choosing this length, we can avoid bias(unavoidable) to the model to be trained. Meanwhile, a smaller length would save us a lot of computing resources.

The max length is 73 which is not a big number for our model. Thus, we decided to set 73 as the padding length.

Resampling

Training neural networks can be an expensive task. It is normal for the artificial neural networks to get from tens to hundreds of thousands trainable parameters. The word “trainable” is to say these parameters would be constantly updated during the training process. We know artificial neural networks use gradient descent and backpropagation to implement the “learning” procedure. During this procedure, the most time-consuming part is back propagation. Given a dataset with 3 million samples, we can hardly train the model on all data every time in the experiments. By the concern of computational capability, we used smaller dataset on the training in experiments to validate our assumptions. The way to acquire smaller datasets are critical, which can directly influence the 合理性 of experiments. It is deductive, from specific to general.

In our experiments, we applied downsampling to obtain smaller dataset. The downsampled dataset would have the same composition. When downsampling, we would randomly choose samples from the old set. The number we choose from each class would be calculated, promising that proportion of each class in the new dataset is as same as it was in the old one. And because our data is balanced, the resampled dataset would be balanced as well.

3.6.2 LSTM (Long Short-Term Memory Neural Network)

Long short-term memory (LSTM) neural network is a special recurrent neural network composed of LSTM units. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on sequences data, it can learn from context.

In our project, we were thinking LSTM should be a good shot compared with CNN. We firstly tried a two-layer with 128 neurons for each layer LSTM model. There were too many parameters in that model each epoch consumed near two hours. The first-round training accuracies reached 98% without overfitting. Thus, we thought maybe we could try a smaller model which will allow us train faster. So we tried a one-layer with 128 neurons LSTM model. The training time for each epoch was correspondingly reduced. In the end we targeted with a one-layer 64-neuron LSTM. With this model we could obtain a balance between the fitting capability and the training speed.
We chose binary cross-entropy as loss function. Followed by the recommendation from Dr. Geoffrey Hinton for his CSC321 class at Toronto University, we chose RMSProp as the optimizer which is recommended as a good choice of RNN models. Last, because this is a classification task, we naturally choose accuracy, sometimes confusion matrix, as our metrics.

3.6.3 CNN (Convolutional Neural Network)

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Typically, CNN is used on image/audio data. It plays a vital role in cognitive computing like image/voice recognition. But there are some approaches to text classification that use CNN as the classifier.
At the first place, we were inspired by the VGG model. We created the first CNN model with a VGG-ish architecture. It soon turned out 2D CNN is not suitable for our task and data. We then convert the model to simple one-dimensional CNN model for our domain data is a 1D vector after character-to-sequence embedding. In our experiments, we found that CNN models overfit easily. There are three main methods we applied to reduce overfitting: Using dropout layer; Adding regularizers; Providing more data.

3.7 Bias and Variance Trade-off

During training a neural network model, we always faced difficulties on fine-tuning. There are so many configuration options for us. With which specific configuration that returns the best model becomes the most confusing decision. This is why we use bias and variance trade-off as the guide during fine-tuning models. Bias refers to the error rate of the neural network model trained on the training set when the training set is large enough. It reflects the capability of your model, the lower the bias you get, the better the model is. On the other hand, the variance is used to measure how much worse the model performs on the testing set compared to on the training set. Big variance indicates the serious overfitting the model has met. The ideal model is the one with both small bias and variance. In other words, well-fitted models with high accuracy are those perfect model we want in this project.
In our experiments, we faced high bias as well as high variance. Techniques were taken to avoid these situations. To avoid high bias, five ways are recommended:

1. Enlarge the size of your model.
2. This means you can increase the number of neurons in layers. This will directly improve the learning ability of your neural network for more trainable parameters.
3. Improve the quality of input features.
4. Many times, the reason that your model doesn't work well may not come from the model itself. It is probably because your features lack abilities to represent your data well. For example, in our project, we choose character-level embedding to represent data.
5. Eliminate regularization
6. Some models add regularization method like l2 regularizer and dropout layer, to prevent overfitting. If the regularization is too much, it will directly lead to the lack of capability of learning. At this time, trying to reduce regularization might be a good choice.
7. Modify the architecture
8. High bias may also come from the simplicity of your model architecture. You can try a larger number of layers or layer with more sophisticated functionalities. But the modification on architecture can be difficult. You should think carefully before changing the architecture.
9. Train the model on more data

10. If more trainable data available, try to use more data to train the model. It remarkably reduced the bias in our experiments.

For reducing variance, there are also a few techniques we used in our experiments and would like to share:

1. Decrease model size
2. High variance means overfitting. The model has too many parameters which allow it to learn perfectly from the training data. Thus the training accuracies look great, yet when it comes to testing data, whole new data that never seen before by the model, the true, not that good, accuracies become what you get. Opposite to what we mentioned above, at this time, you may decrease the number of neurons inside your model to reduce its learning ability.
3. Add regularization
4. Another useful method is adding regularization. In our experiments, we applied l2 regularizers on Conv2D layers.

More training data

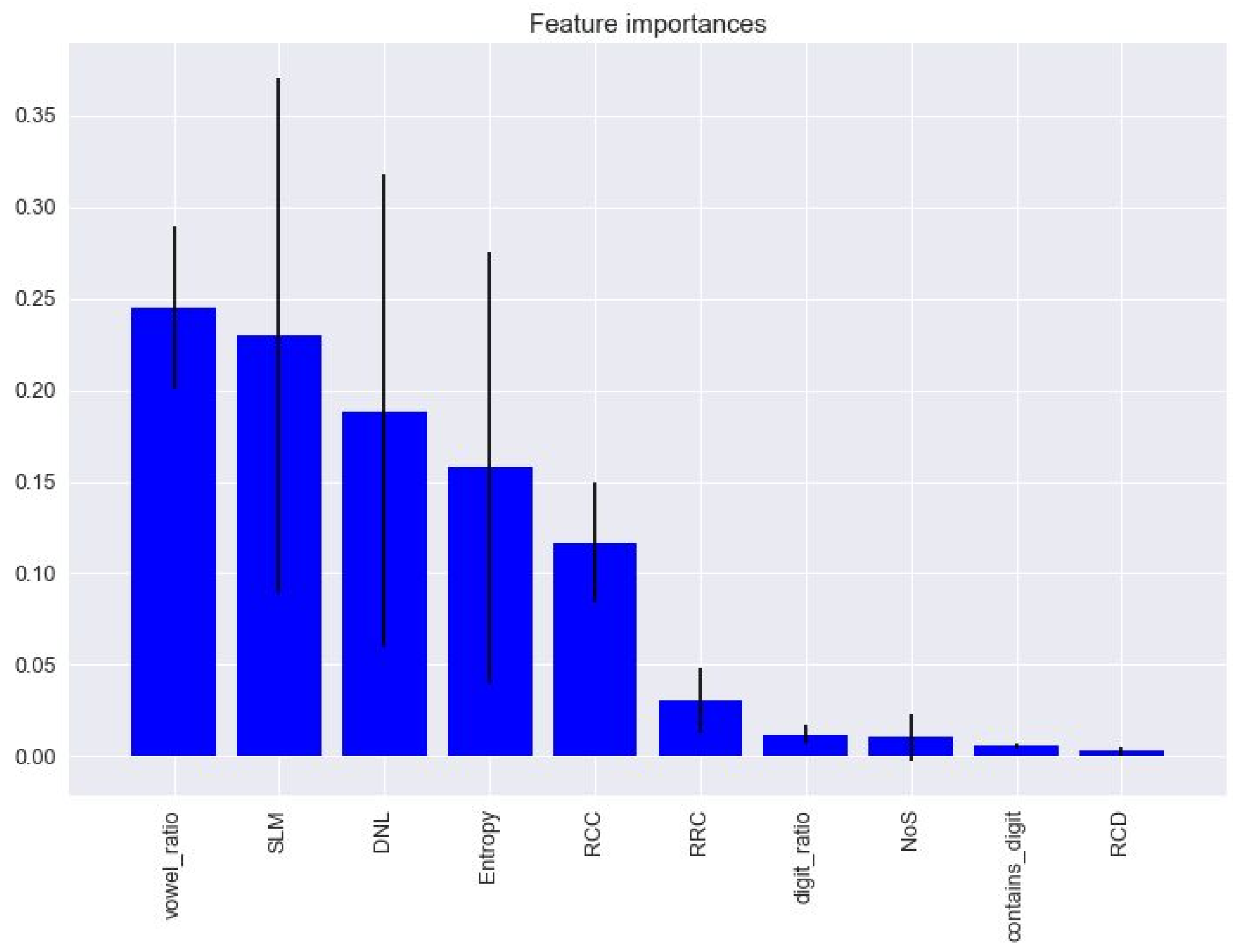
Last but not least, you can also try to add more data to your training set. It really helps!

4. Results

4.1 Random Forest

After applying the Random Forest model, training dataset gets 92.23% accuracy and test dataset get 91.65% accuracy. One advantage Random Forest model is that the time for building and fitting a model is not too much. So, it can finish its training part quickly.

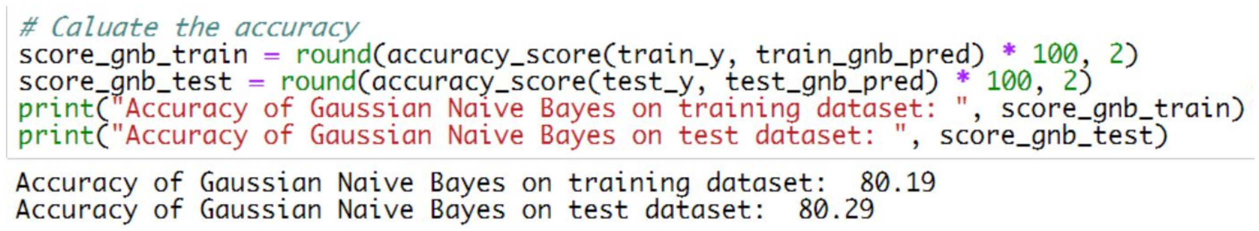
Besides the performance of Random Forest model, the graph of importance for each attribute in the model has also been generated. From the figure below, vowel_ratio, SLM, DNL play the most important roles in the final decision.



Figure, the importance of each attribute in the Random Forest model

4.2 Naive Bayes

The accuracy of Gaussian Naive Bayes model:

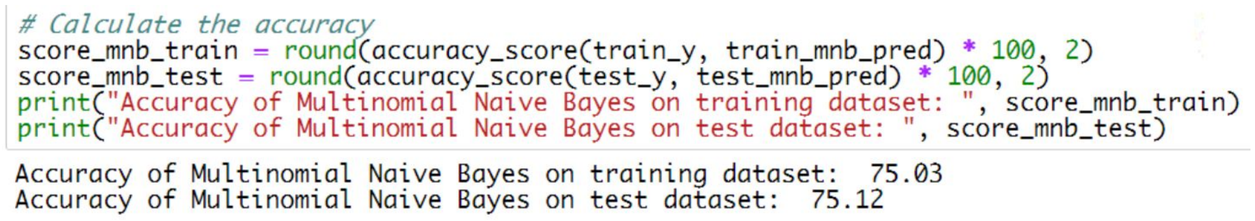


```
# Caluate the accuracy
score_gnb_train = round(accuracy_score(train_y, train_gnb_pred) * 100, 2)
score_gnb_test = round(accuracy_score(test_y, test_gnb_pred) * 100, 2)
print("Accuracy of Gaussian Naive Bayes on training dataset: ", score_gnb_train)
print("Accuracy of Gaussian Naive Bayes on test dataset: ", score_gnb_test)
```

```
Accuracy of Gaussian Naive Bayes on training dataset:  80.19
Accuracy of Gaussian Naive Bayes on test dataset:  80.29
```

Figure Gaussian Naive Bayes model

The accuracy of Multinomial Naive Bayes model:

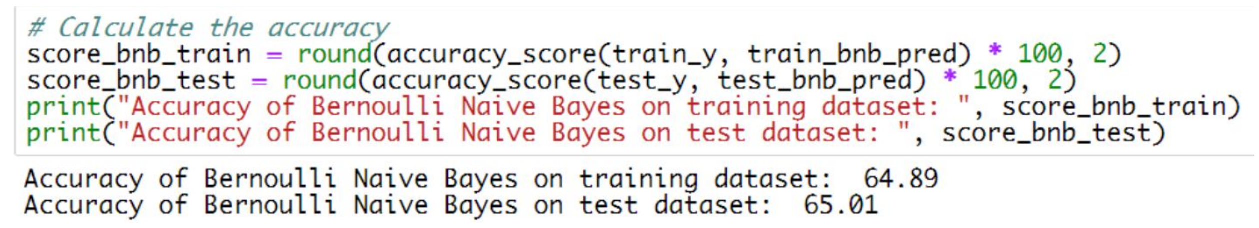


```
# Calculate the accuracy
score_mnb_train = round(accuracy_score(train_y, train_mnb_pred) * 100, 2)
score_mnb_test = round(accuracy_score(test_y, test_mnb_pred) * 100, 2)
print("Accuracy of Multinomial Naive Bayes on training dataset: ", score_mnb_train)
print("Accuracy of Multinomial Naive Bayes on test dataset: ", score_mnb_test)
```

```
Accuracy of Multinomial Naive Bayes on training dataset:  75.03
Accuracy of Multinomial Naive Bayes on test dataset:  75.12
```

Figure Multinomial Naive Bayes model

The accuracy of Bernoulli Naive Bayes model:



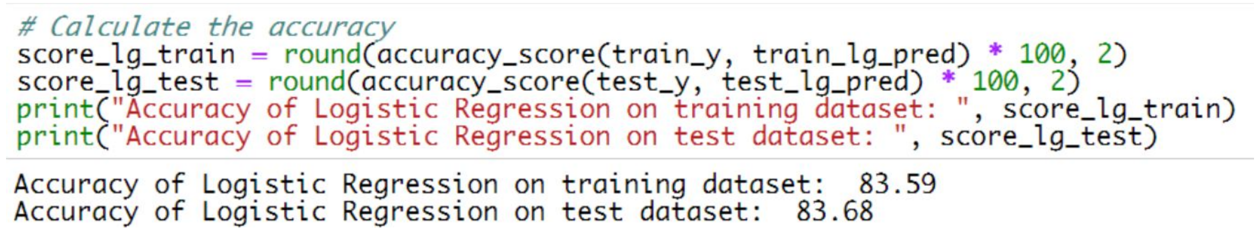
```
# Calculate the accuracy
score_bnb_train = round(accuracy_score(train_y, train_bnb_pred) * 100, 2)
score_bnb_test = round(accuracy_score(test_y, test_bnb_pred) * 100, 2)
print("Accuracy of Bernoulli Naive Bayes on training dataset: ", score_bnb_train)
print("Accuracy of Bernoulli Naive Bayes on test dataset: ", score_bnb_test)
```

```
Accuracy of Bernoulli Naive Bayes on training dataset:  64.89
Accuracy of Bernoulli Naive Bayes on test dataset:  65.01
```

Figure Bernoulli Naive Bayes model

4.3 Logistic Regression

The Logistic Regression model reached 83.68% accuracy on the test dataset.



```
# Calculate the accuracy
score_lg_train = round(accuracy_score(train_y, train_lg_pred) * 100, 2)
score_lg_test = round(accuracy_score(test_y, test_lg_pred) * 100, 2)
print("Accuracy of Logistic Regression on training dataset: ", score_lg_train)
print("Accuracy of Logistic Regression on test dataset: ", score_lg_test)
```

```
Accuracy of Logistic Regression on training dataset:  83.59
Accuracy of Logistic Regression on test dataset:  83.68
```

Figure Logistic Regression model

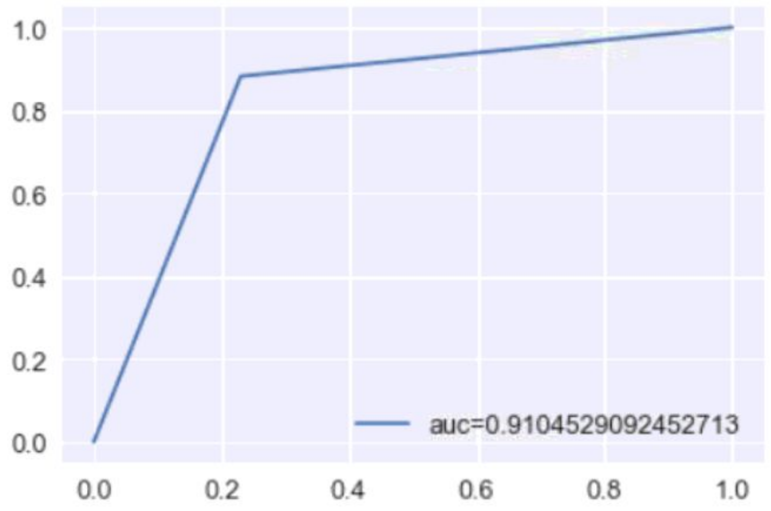
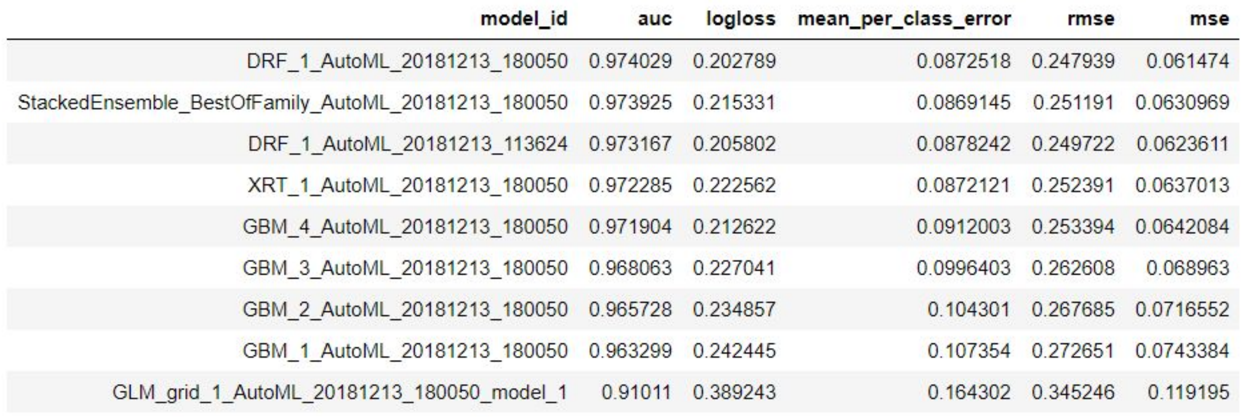


Figure ROC of Logistic Regression

4.4 H2O

As for the project, the size of the whole dataset is about 2.5 million. The first time we run H2O Auto Machine Learning, H2O took about 40 minutes to find the best model with the accuracy of 95%. For this training, only using half of the whole dataset. By putting the whole 2.5-million dataset into H2O Auto Machine Learning function, it takes about 2 hours to finish the fitting and predicting part. Luckily, the latest training reached the accuracy of 97.40%.

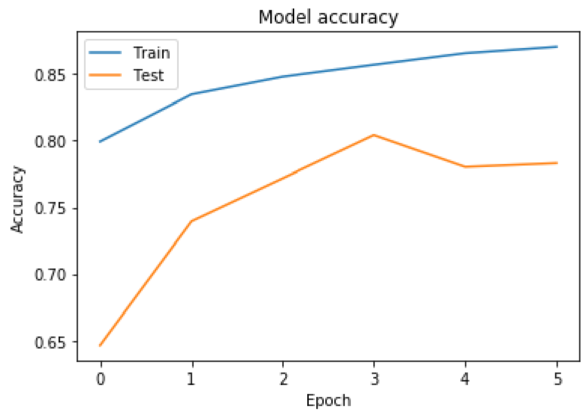


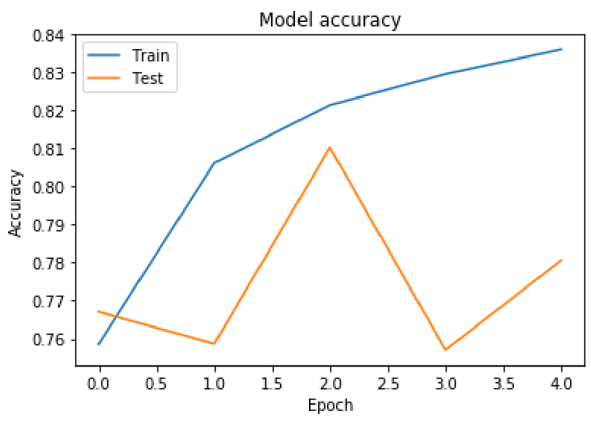
model_id	auc	logloss	mean_per_class_error	rmse	mse
DRF_1_AutoML_20181213_180050	0.974029	0.202789	0.0872518	0.247939	0.061474
StackedEnsemble_BestOfFamily_AutoML_20181213_180050	0.973925	0.215331	0.0869145	0.251191	0.0630969
DRF_1_AutoML_20181213_113624	0.973167	0.205802	0.0878242	0.249722	0.0623611
XRT_1_AutoML_20181213_180050	0.972285	0.222562	0.0872121	0.252391	0.0637013
GBM_4_AutoML_20181213_180050	0.971904	0.212622	0.0912003	0.253394	0.0642084
GBM_3_AutoML_20181213_180050	0.968063	0.227041	0.0996403	0.262608	0.068963
GBM_2_AutoML_20181213_180050	0.965728	0.234857	0.104301	0.267685	0.0716552
GBM_1_AutoML_20181213_180050	0.963299	0.242445	0.107354	0.272651	0.0743384
GLM_grid_1_AutoML_20181213_180050_model_1	0.91011	0.389243	0.164302	0.345246	0.119195

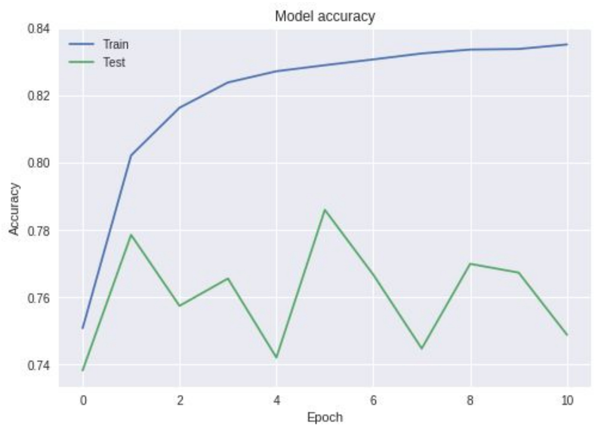
Figure the leaderboard of H2O Auto Machine Learning

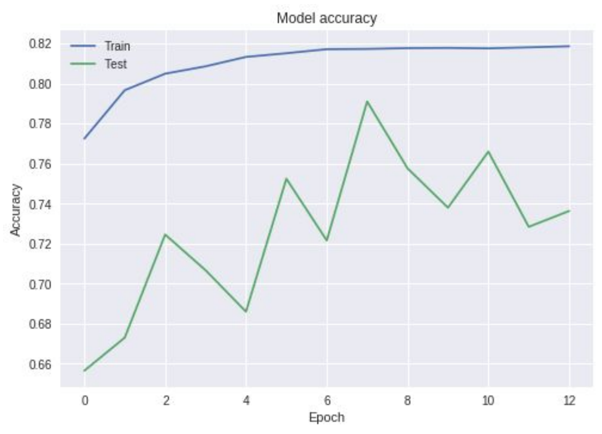
4.5 CNN

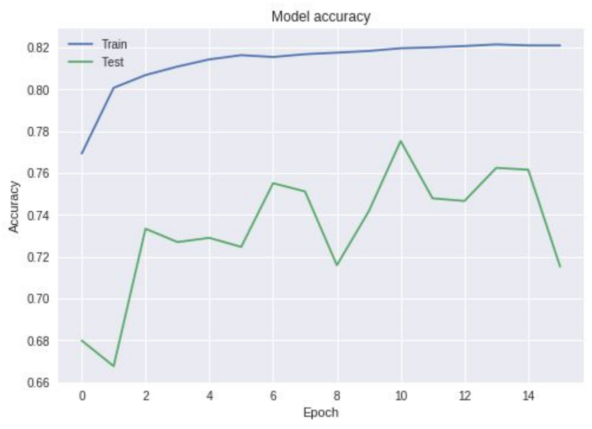
We tried many CNN models with different structures and configurations. Before we have explained the reason we think 1D CNN is better than 2D. However, no matter which, the variances are big and the training was not stable. The best CNN model only gave us an accuracy around 79%. We soon decided to abandon CNN models and went with LSTM models for the deep learning part.

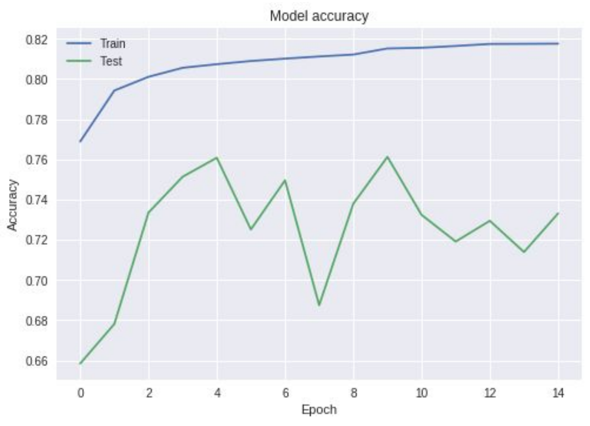


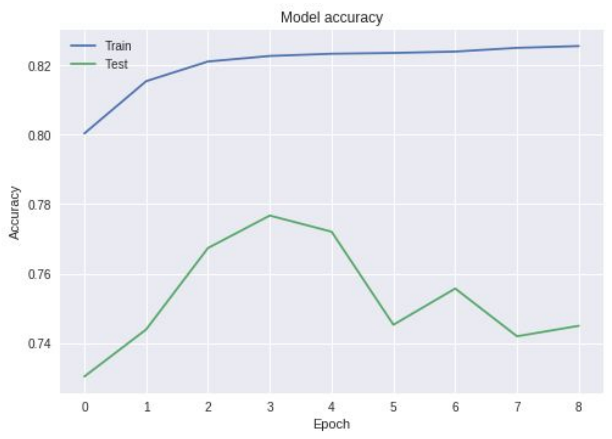








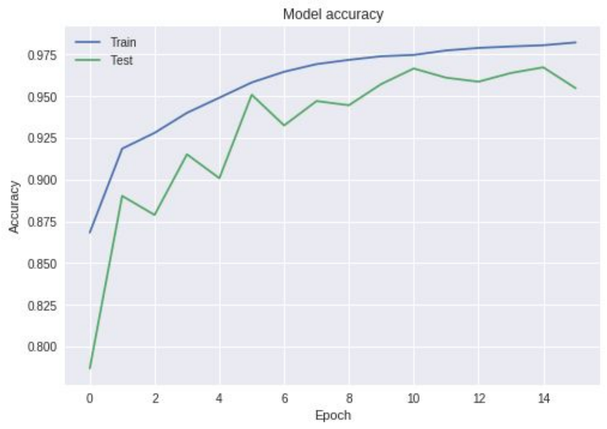


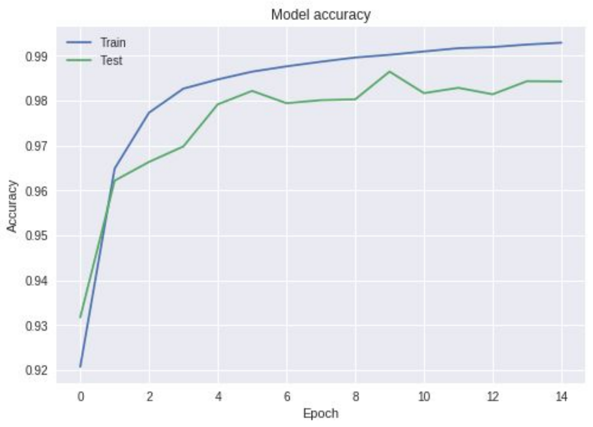


4.6 LSTM

The best accuracy we got from LSTM model is 98.56%. This accuracy came from a simple one-layer LSTM with 2 million data (1m benign data and 1m DGA domains). LSTM models have strong capabilities to learn from our data. We started with trying tow-layer LSTM, but we soon found one-layer has the same learning capability. When two neural networks get similar performance, we decided to use the simple one by following the principle of Occam's Razor.

Here are two typical training curves for LSTM models.





5. Discussion

By summarising the work we have done, the graph of each model’s performance has been generated. It can be known that Long Short-Term Memory neural network and H2O model have the best performance in DGA Detection. From the process that building our model, LSTM has the best performance but it will need a lot of time to train. However, for the H2O part, it only need a few time to train the model and it can classify a domain as soon as possible.

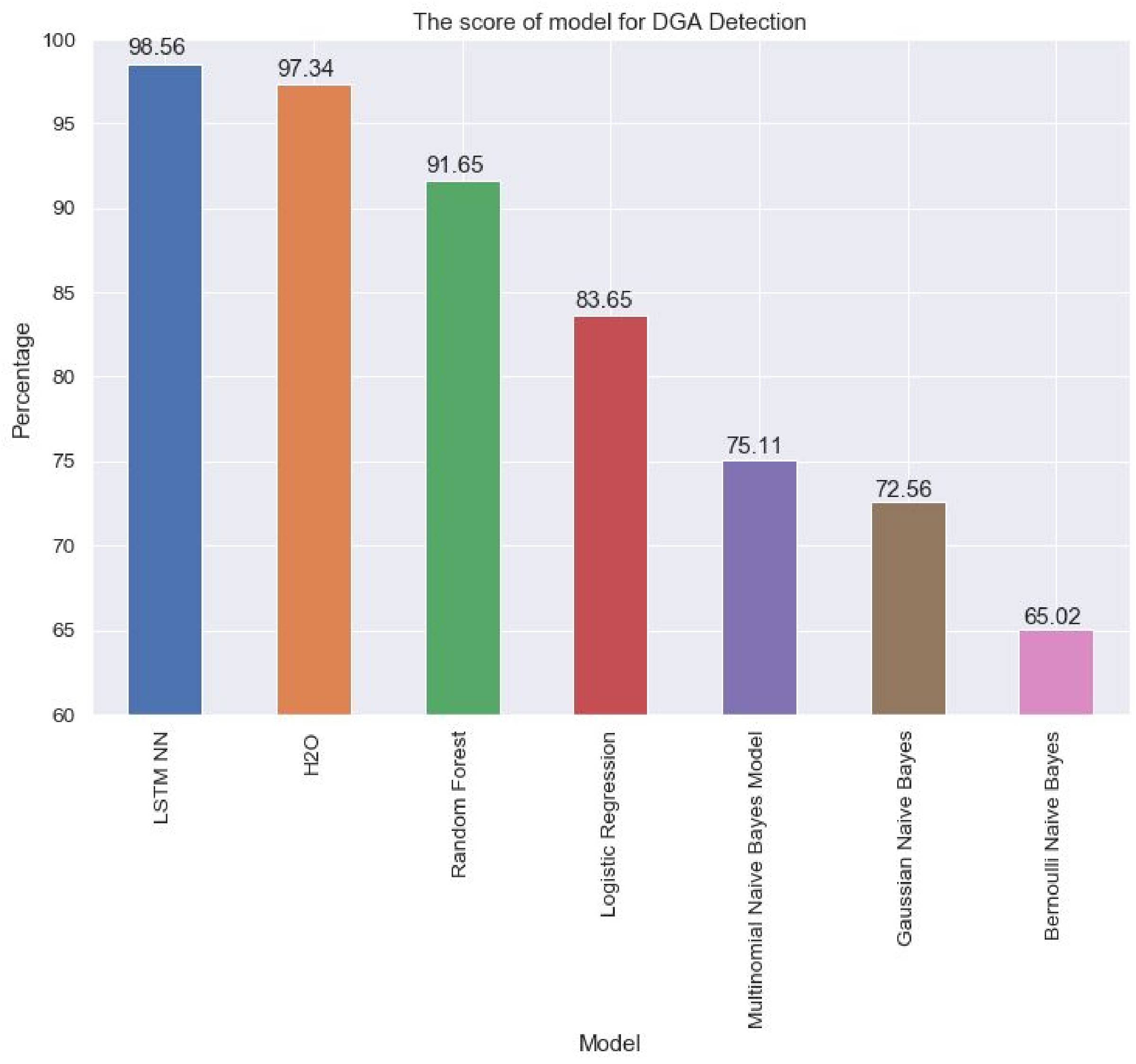


Figure The performance of each model in DGA Detection

References

[1] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. (2012, August). From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In USENIX security symposium (Vol. 12).

[2] AWS | Alexa Top Sites-Up-to-date lists of the top sites on the web. (n.d.). Retrieved from http s://aws. amazon.com/alexa-top-sites/

[3] Domain generation algorithm. (n.d.). Retrieved from https://www.wikiwand.com/en/Domai n_ generation_algorithm

[4] G. (2018, November 05). Google-research/bert. Retrieved from https://github.com/google-res earch/bert

[5] H2O – Data Resource Portal. (n.d.). Retrieved from https://www.northeastern.edu/datares o urces/h2o

[6] H. A., & J. W. (2018, February 22). Using Deep Learning to Detect DGAs. Retrieved from h t tps://www.endgame.com/blog/technical-blog/using-deep-learning-detect-dgas

[7] Koehrsen, W. (2018, June 02). Automated Feature Engineering in Python – Towards Data Science. Retrieved from https://towardsdatascience.com/automated-feature-engineeri ng-in-pyt hon-99baf11cc219

[8] Plohmann, D., Yakdan, K., Klatt, M., Bader, J., & Gerhards-Padilla, E. (2016, August). A Comprehensive Measurement Study of Domain Generating Malware. In USENIX Security Symposium (pp. 263-278).

[9] Schiavoni, S., Maggi, F., Cavallaro, L., & Zanero, S. (2014, July). Phoenix: DGA-based botnet tracking and intelligence. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 192-211). Springer, Cham.

[10] Schüppen, S., Teubert, D., Herrmann, P., Meyer, U., & Sch, S. (2018, August). FANCI: feature-based automated NXDomain classification and intelligence. In Proceedings of the 27th USENIX Conference on Security Symposium (pp. 1165-1181). USENIX Association.

[11] Tran, D., Mac, H., Tong, V., Tran, H. A., & Nguyen, L. G. (2018). A LSTM based framework for handling multiclass imbalance in DGA botnet detection. Neurocomputing, 275, 2401-2413.

[12] Woodbridge, J., Anderson, H., Ahuja, A., & Grant, D. (2016). Predicting Domain Generation Algorithms with Long Short-Term Memory Networks.

[13] Yu, B., Pan, J., Hu, J., Nascimento, A., & De Cock, M. (2018). Character Level Based Detection of DGA Domain Names.