# POLITECNICO DI TORINO

**Master's Degree in Computer Engineering**


**Master's Degree Thesis**

# DGA Detection with Big Data approaches

**Supervisors**

**Prof. Paolo GARZA**

**Company Tutor**

**Emanuele GALLO**

**Candidate**

**Luigi DE LUCA**

**Academic Year 2022-2023**

**Abstract**

Domain generation algorithms (DGA) are algorithms that are present in various families of malware that are used to periodically generate a large number of domain names that can be used to communicate with their command and control servers. Domain Generation Algorithms have quickly become the main method used by the attackers to remotely communicate with the malicious tools that they have created. They no longer make use of hard-coded domain name lists and IP addresses, which are useless once they have been blocked. DGAs, compared to the previous methods, are easy to implement, difficult to block, and may be impossible to predict in advance. The main part of a Domain Generation Algorithm is the domain generator, that can be set as a random string of characters, a concatenation of random words taken from a dictionary, a constant part followed by a changing suffix, a constant part preceded by a changing prefix and so on. The purpose of this thesis project is to address and study DGA detection solutions, analyzing and studying the characteristics of the DGA domain names and trying to create a model that can distinguish between legit and DGA-based domain names. Two different approaches have been analyzed and experimented with which it has been tried to identify DGA domain names: one of supervised machine learning type, based on feature extraction, and one based on deep learning models, based on text classification. The traditional Machine Learning classifiers used are the Random Forest and the XG-Boost, while the two Deep Learning models are based on a Neural Network (NN): the first with a Long Short-Term Memory (LSTM), the second with a Bidirectional LSTM. The dataset used for the validation of the described models is made of real domain names and DGA-based ones, and is divided in training and testing set for the models evaluation. The validation is made in two ways: in the first one there is a random split of the dataset in training and testing set, in the second one a different set of DGA families is used as testing set in order to simulate the case in which the model encounters something that it has never seen, so new kind of DGA families. The models are trained with the training set and evaluated with the testing set, and the results are analyzed with different metrics: the accuracy and the values of True Positives, False Positives, True Negatives and False Negatives are the most important ones. Based on the validation made with the two simulations, the results obtained and the performances in terms of time, it turns out that the best solution is the one based on the XG-Boost Classifier with feature extraction. To be precise, in the second validation, that is the most important one because it simulates a real-word situation, the overall Accuracy of the XG-Boost Classifier is around 92%, the True Positive percentage is slightly less than 89% and the False Positive one is less than 4%. So, XG-Boost is chosen

mainly because of its robustness that it has showed while it encounters algorithms it has never seen before.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction: DGA Detection

Domain generation algorithms (DGA) are algorithms that are present in various families of malware that are used to periodically generate a large number of domain names that can be used to communicate with their command and control servers. The large number of potential meeting points makes it difficult for law enforcement to effectively shut down botnets. This happens because the infected computers will attempt to contact some of these domain names every day in order to receive updates and/or commands. The malware codes make use of public-key cryptography in order to make it unfeasible for law enforcement and other actors to mimic commands from the malware controllers, because some worms will automatically reject the updates that are not signed by the malware controllers.

Domain Generation Algorithms have quickly become the main method used by the attackers to remotely communicate with the malicious tools that they have created. Adversaries no longer make use of hard-coded domain name lists and IP addresses, because they become useless once they have been blocked. DGAs, compared to the previous methods, are easy to implement, difficult to block, and may be impossible to predict in advance. The other advantage is that it is possible to quickly modify them if the previously used algorithm becomes well-known.

A Domain Generation Algorithm typically is based upon three components:

1. A time-sensitive "seed"
2. A domain "body" generator that makes use of the seed

3. A set of top-level domains (the so called TLDs)

Often, the seed could be simply the current date taken in a standard format. The domain body generator is the core of a Domain Generation Algorithm, and can be set as a random string of characters, a concatenation of random words taken from a dictionary, a constant part followed by a changing suffix, a constant part preceded by a changing prefix and so on. The set of Top Level Domains must contain real-world values, and they determine under which Web entities the generated domains are registered.

DGA domains are extensively used by many kinds of malware to communicate to the Command and Control servers. Before the advent of Domain Generation Algorithms, the major part of the malicious programs used hard-coded lists of IP addresses or domain names. DGA is much harder to block, with respect to the latter solutions, by anti-malware software or network administrators since it is almost impossible to predict the next place where commands may come from. It is of crucial importance for businesses to be able to detect network requests made to DGA domain names on the first phases of malware spread in order to minimize the number of infected machines and to reduce the recovery cost.

Command and Control (C&C) servers are used by attackers to operate communications. In order to perform many kind of attacks, the attackers usually make use of the Domain Generation Algorithm (DGA), thanks to which they can confirm meeting points to their C&C servers by generating a large number of network locations, as described in a simplified way in Figure 1.1.

So, they take advantage of DGA in order to automatically create a large set of pseudo-random domain names, and then they choose one ore more of them to resolve the IP address to communicate with the C&C server, improving the reliability and robustness of the communication between the malware and the C&C server itself. The detection of DGA-based domain names is one of the most important technologies for the command and control communication detection.

Common defences against malicious DGA domain names include blacklists, Random Forest Classifiers and clustering techniques. When the lists are well maintained and updated, and there is a careful choice of the features, these methods

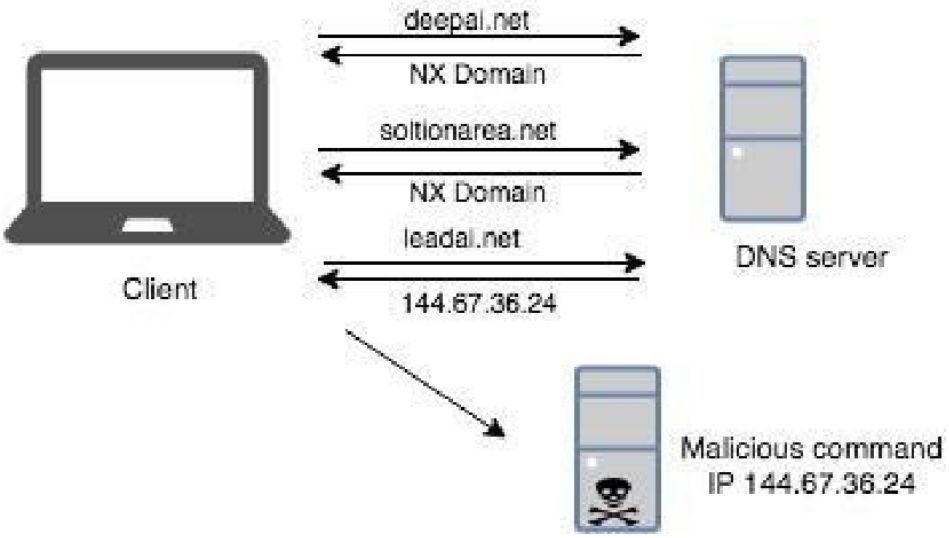



**Figure 1.1:** Example of DGA domains utilization to communicate with C&C

have acceptable efficacy. However, both blacklists and these classification models present serious limitations: first of all, they rely on hand-picked and pre-defined features that are also time-consuming to develop, they do not have the ability to generalize because based on the few manual features implemented, and require continuous expert maintenance. There is the necessity to find more comprehensive tactics able to detect the new Domain Generation Algorithm families that come from network-based malware.

The purpose of this thesis project is to address and study DGA detection solutions, analyzing and studying the characteristics of the DGA domain names and trying to create a model that can distinguish between legit and DGA-based domain names.

For the work two different approaches have been analyzed and experimented with which it has been tried to identify DGA domain names: one of *supervised machine learning* type, based on feature extraction, creation of the best feature set, application of anomaly detection algorithms over the domain names, and one based on *deep learning* models.

In the supervised machine learning approach with feature extraction, the basic

idea is to create the best set of features of the domain names to train our classifier in order to distinguish between legit and DGA domain names.

The second solution is based on deep learning models, training and use them in order to do text classification on the subdomain string and distinguish legit domain names and DGA ones.

# Chapter 2

# Related Work

In recent years, the major part of malware families have started a different kind of approach in order to communicate with their remote servers.

Instead of using hard-coded domain name lists and IP addresses, these malware families make use of Domain Generation Algorithms (DGAs). DGAs are a class of algorithms that takes a seed as input and, thanks to some logic, that is the core of the algorithm, returns a string and then appends a top level domain (TLD).

There are various kind of DGA techniques that vary in the core logic and in complexity, in order to prevent the detection of malicious domain names. For example, the newest DGA families try to simulate the composition of the real domain names, the so called word-list-based Domain Generation Algorithms, and make the detection more difficult.

Nowadays, the problem of identifying malicious DGA domain names has become more and more important and has received a lot of attention.

One of the first detection techniques, is the N-Grams approach, described in [1], that explains that, in the DGA detection, one common method is to split the domain names into n-grams and then apply a kind of frequency analysis. It is a very effective method in terms of performance, but the accuracy is average. In this article, there is also the description of a simple LSTM model in order to address the DGA detection problem, which seems very promising.

Another approach, with the code example present in [2], makes use of the Spark library MLib [3]. This method is based on a feature engineering process, in which a large number of features, that represent the characteristics of the domain names, has been extracted and used in order to train a classifier in the distinction between real and DGA-based domain names.

In their paper, Ren, F., Jiang, Z., Wang, X. et al. [4] propose a DGA detection method based on Deep Learning models. The idea is to build a deep learning framework, called ATT-CNN-BiLSTM, in order to identify malicious domain names. First of all, a Convolutional Neural Network (CNN) and a Bidirectional Long Short-Term Memory (BiLSTM) neural network layer are implemented in order to extract the features of the domain names; then, an Attention layer is used in order to allocate the corresponding weight of the extracted deep information from the domain names. Finally, the different weights of features in domain names are the inputs of the output layer, that completes the tasks of detection and classification.

Highnam, K., Puzio, D., Luo, S. et al. [5] propose an innovative deep learning method in order to address the DGA detection problem. They created a novel hybrid neural network, the so called Bilbo the "bagging" model, that tries to analyze domain names and to score the likelihood that they have been generated by DGA algorithms and so can be potentially malicious. Bilbo makes a parallel usage of a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) Network for the DGA detection problem.

In their paper, Sciancalepore, S. and Namgung, J. and Son, S. and Moon, Y. [6] start from a simple Long Short-Term Memory (LSTM) based deep learning model, in order to build an efficient DGA domain name detection method that is based on a Bidirectional LSTM (BiLSTM), which has the advantage to learn bidirectional information as opposed to unidirectional information that can be learned by a simple LSTM. They also try to maximize the detection performance with a Convolutional Neural Network, creating a (CNN)+BiLSTM ensemble model using an Attention mechanism, which allows the model to learn both local and global information in a domain sequence.

All of these articles, papers and code examples have been an useful starting point in the analysis of the DGA detection problem and in the development of the approaches and models used in this thesis work.

# Chapter 3

# First Approach: Feature Extraction Method

In recent years, the research of identifying DGA domain has received extensive attention. The task of confirming whether or not a domain name is generated by a DGA is a crucial step of malware defenses.

This thesis work, therefore, aims to study and experiment, with Big Data techniques, the problem of DGA detection, in other words, the classification of a domain name as legit or not. In fact, it is possible to try to look at the topic of Big Data not as a possible threat, but as an opportunity (or tool) to build effective security methods.

The objective of the addressed problem is to study the differences between real domain names and DGA-based ones and then to find the best possible model that is able to make this kind of distinction, so if a domain is malicious or not. Various models will be trained and tested in order to find the one that will give the best results.

The first approach used in this experiment is based on the analysis of the domain names, finding the best set of features that can help a classifier to distinguish between legit and DGA domain names.

## 3.1 Input Data

The dataset used for this thesis work is composed by a list of benign domains and a list of DGA domain names created by different families of algorithms.

- The list of the benign domains is taken from the Alexa's top 1M[7].
- The list of the DGA domains is taken from Netlab 360[8].

In the Alexa's top 1M, there are about 1 million of legit domains that are the most popular. The fields in this set are only ad ID and the domain name.

In the set of DGA domains, there are many domain names generated by different DGA families. The fields in this file are:

1. Family, the family name of the algorithm that has generated the domain name
2. Domain, the domain name
3. Start, the beginning of valid time of the domain
4. End, the end of valid time of the domain

## 3.2 N-Grams Approach

The first idea in the analysis of the domain names is to study the composition of the string itself, and to do so, there is the utilization of the n-grams.

N-grams are continuous sequences of characters, symbols, words or tokens in a document. In technical terms, they can be defined as the contiguous sequences of items in a document, as shown in Figure 3.1. The value of N can be any positive integers starting from 1, although usually large N are not considered because those n-grams rarely appears in many different words, sentences or documents.

They come into play when we deal with text data, so in the analysis of the domain names, they can be a useful starting point. When performing machine

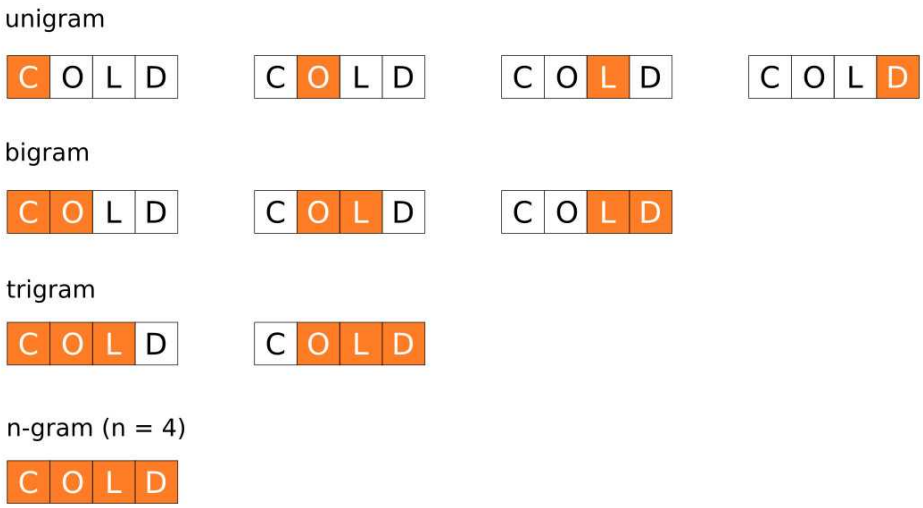



**Figure 3.1:** Example of N-Grams at character level

learning tasks related to Natural Language Processing (NLP) problems, n-grams are generated starting from input sentences. For example, in text classification tasks, in addition to use each individual characteristic found in the text, 2-grams or 3-grams can be added as features.

This is the way n-grams will be used in domain names analysis: splitting the domain name into N-grams followed by frequency analysis.

The input data from which the n-grams are computed are two: the domain names themselves and an English words list. We compute n-grams for every Alexa domain and also on a dictionary of words. After computing these two different types of n-grams, we compute n-gram matches for all the domain names, performing a kind of frequency analysis of the n-grams in every domain name, and add these two values as features.

Only the study and analysis of n-grams is not enough to perform the classification of the domain names and resolve the problem of the DGA detection. So, we need to find other features in order to go ahead in the work.

## 3.3 Feature Engineering

With the dataset available, the first task performed is the Feature Engineering: the purpose is to extract useful information for the algorithm, starting from raw domain names, taken as they are made available.

This procedure is done mainly to find a set of characteristics of the domain names, considered as simple strings, that can help in the job of distinguish between real and DGA-based domains. This is done by taking into consideration the "raw" domains in such a way as to create "quantitative" features, since these better describe them and better highlight any anomalies.

Thus, the feature engineering operation is done for each domain name: in this way the information collected will represent every domain and, by correlating them, the algorithm will be able to better classify them in the two classes.

The feature extraction process has been a very long operation, in which many features were added and computed for every domain name, trying to find the best possible set of features that helps the algorithm to classify the domains. After many trials and attempts, some of the initial features were removed because they were useless for the algorithm in this classification problem, and only the relevant ones remained.

The initial features removed from the final set were removed because the classifiers did not use them in order to label the domains. These removed features were:

- NoS, representing the number of subdomains of the domain name.
- HVTLD, representing a flag that tells if the domain has a valid TLD (1) or not (0).
- CTS, representing a flag that tells if the domain has a TLD as subdomain (1) or not (0).
- RRC, Ratio of Repeated Characters, representing the percentage of repeated characters contained in the domain.

So, the following set of features was created:

- DNL, representing the length of the domain name.
- SLM, representing the subdomain length mean, considering the subdomains that compose the domain.
- contains_digit, representing a flag that tells if the domain has a digit (1) or not (0).
- digit_ratio, representing the percentage of digits contained in the domain.
- vowel_ratio, representing the percentage of vowels contained in the domain.
- RCC, Ratio of Consecutive Consonants, representing the percentage of consecutive consonants contained in the domain.
- Entropy, representing the entropy of the domain, in other words, the measure of uncertainty, because the more random a string is, the higher its calculation of the entropy.
- alexa_grams, representing the n-gram matches of the domain, considering the n-grams computed from the Alexa domains, as explained in section 3.2.
- word_grams, representing the n-gram matches of the domain, considering the n-grams computed from a dictionary of words, as explained in section 3.2.

Thanks to having performed many tests, a curiosity stood out: the major part of the real domain names have a higher value of the Alexa-grams matches with respect to the DGA generated ones. But the relevant thing is that also some DGA-based domain names have a quite high value of the Alexa-grams matches too, comparable to the values of legit domains.

The difference is that, while the real domain names have high values of the Alexa-grams matches feature, the DGA-based ones that have the value of this feature high too, are domains with a very high length, as shown in Figure 3.2. So, in order to distinguish these two situations, another feature was added to the dataset:

- ngrams_slm_ratio, representing the ratio between the Alexa n-grams and the SLM of the domain.

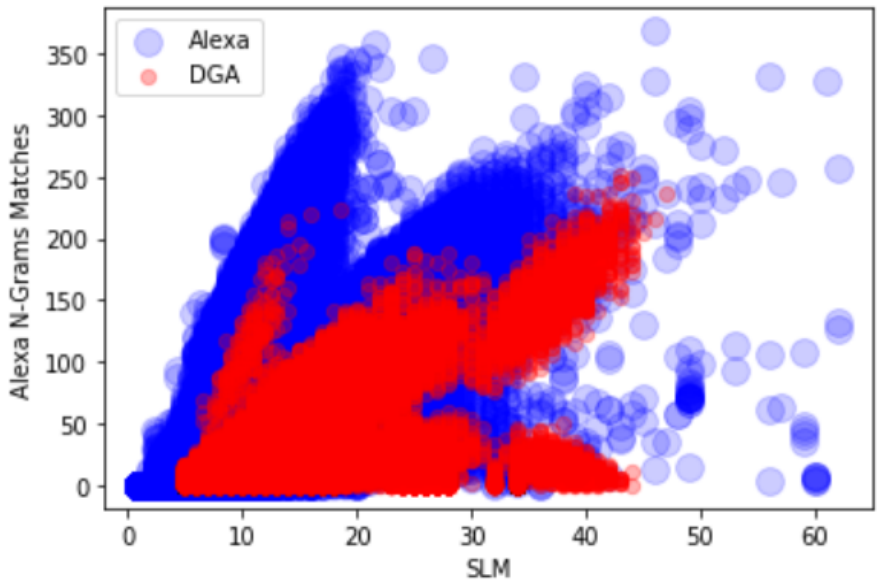



**Figure 3.2:** Correlation between Subdomain Length Mean and Alexa N-Grams Matches

Going deeper in the reasons why these features have been chosen, there is the distribution analysis that can help to understand the choices made in the feature extraction process.

The distribution of the domain name length (DNL) of legit domains, shown in Figure 3.3, and DGA-based ones, shown in Figure 3.4, explains that real domains have shorter length with respect to DGA ones. It is quite impossible to have DGA domains with a very short length, because they are random sequences of characters or of English words, while the real ones have similar length because they are human generated.

The distribution of the subdomain length mean (SLM) of legit domains is described in Figure 3.5, while the SLM of the DGA-based ones is in Figure 3.6. The SLM follows the things said for the domain name length (DNL), in fact real domains have lower SLM and it is a sort of Gaussian distribution, while the DGA ones have, in general, higher values.

The digit ratio distribution of legit domains, described in Figure 3.7, and of

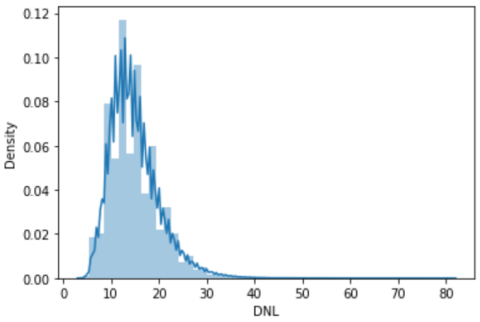



**Figure 3.3:** Domain Length distribution of legit domain names

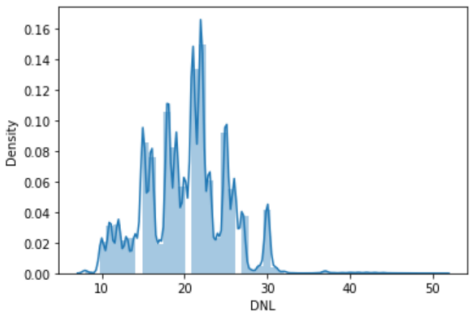



**Figure 3.4:** Domain Length distribution of DGA domain names

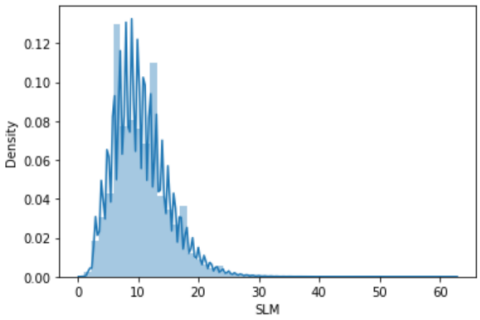



**Figure 3.5:** Subdomain Length Mean distribution of legit domain names

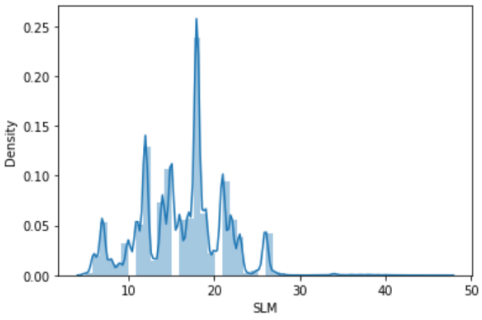



**Figure 3.6:** Subdomain Length Mean distribution of DGA domain names

the DGA-based ones, in Figure 3.8, shows that the major part of the real domain names have no digits, while there is a slightly higher number of DGA domain names that have a higher digit ratio.

The vowel ratio distribution of legit domains is show in Figure 3.9, while the DGA-based domains one is shown in Figure 3.10. This feature tells that, in general, DGA domain names have a lower value of vowel ratio, even if also real ones have not very high values.

The distribution of the Ratio of Consecutive Consonants (RCC) of legit domains is show in Figure 3.11, while RCC of the DGA-based domains is shown in Figure 3.12. These distributions show that real domains have, in general, lower number

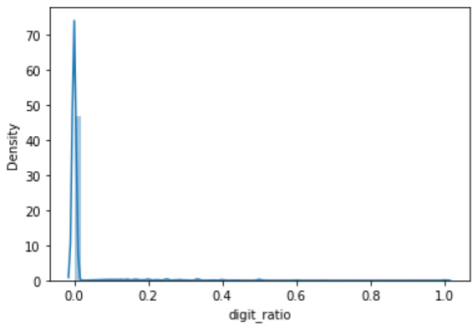



**Figure 3.7:** Digit Ratio distribution of legit domain names

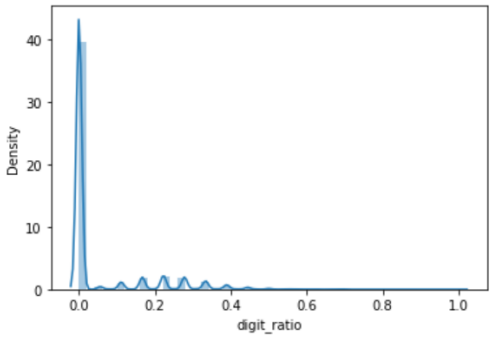



**Figure 3.8:** Digit Ratio distribution of DGA domain names

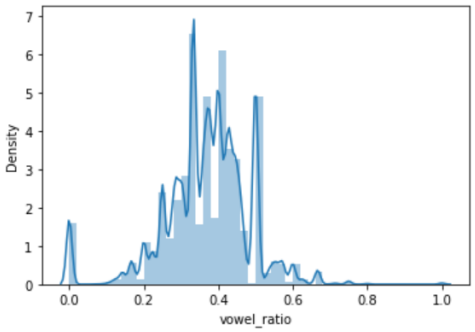



**Figure 3.9:** Vowel Ratio distribution of legit domain names

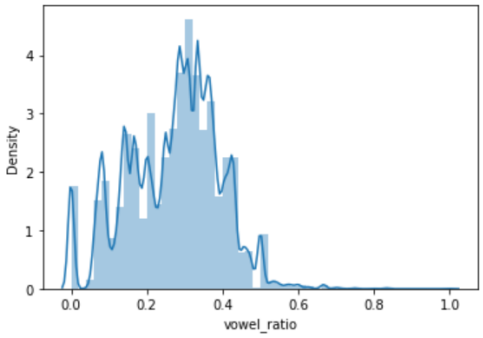



**Figure 3.10:** Vowel Ratio distribution of DGA domain names

of consecutive consonants, while the DGA ones have high values because, in most cases, they are a random sequence of characters.

The Entropy distribution, described in Figure 3.13 for legit domains and in Figure 3.14 for the DGA ones, shows that DGA domains have an entropy value higher than 2, except for few cases. Also legit ones have high values of entropy, but they are more downward-shifted.

The Alexa N-Gram Matches distribution of legit domains is shown in Figure 3.15, while the one of the DGA domains is shown in Figure 3.16. This feature tells that real domain names have, obviously, more Alexa N-Gram matches with respect

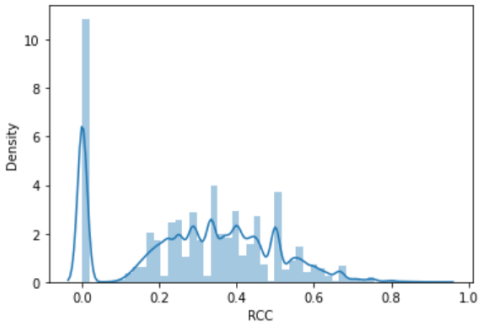



**Figure 3.11:** Ratio of Consecutive Consonants distribution of legit domain names

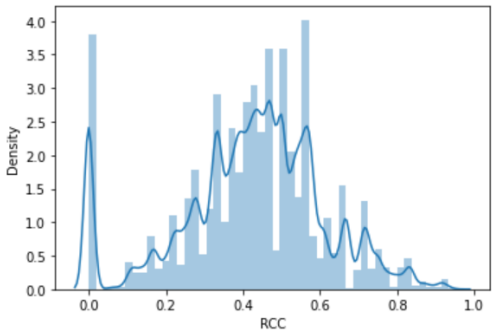



**Figure 3.12:** Ratio of Consecutive Consonants distribution of DGA domain names

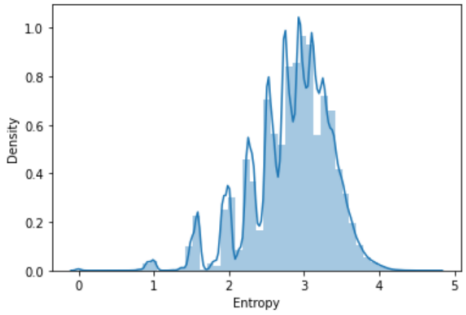



**Figure 3.13:** Entropy distribution of legit domain names

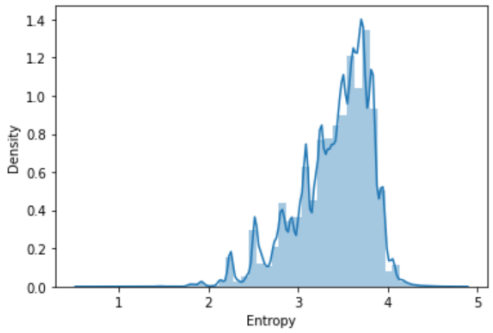



**Figure 3.14:** Entropy distribution of DGA domain names

to the DGA ones, because the latter are randomly generated.

The Word N-Gram Matches distribution of legit domains, described in Figure 3.17, and the one of the DGA domains, described in Figure 3.18, show that the major part of real domain names have high values of this type of matches, while the DGA ones have lower values, except for the word-list-based DGA domain names.

The distribution of the Alexa N-Grams/SLM Ratio is described in Figure 3.19 for the legit domain names and in Figure 3.20 for the DGA ones. It can be seen that the real domains have higher values of this ratio with respect to the DGA

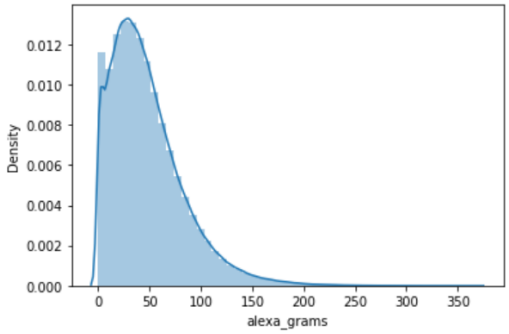



**Figure 3.15:** Alexa N-Grams distribution of legit domain names

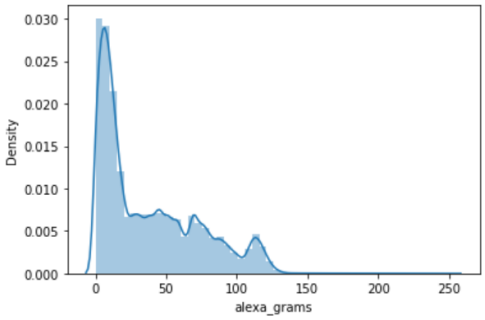



**Figure 3.16:** Alexa N-Grams distribution of DGA domain names

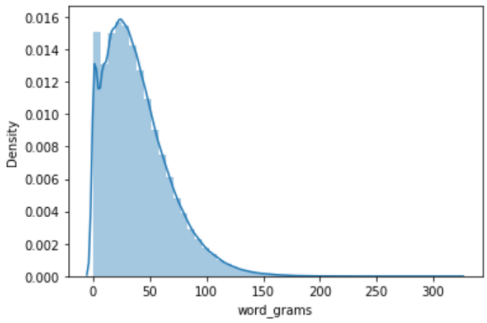



**Figure 3.17:** Word N-Grams distribution of legit domain names

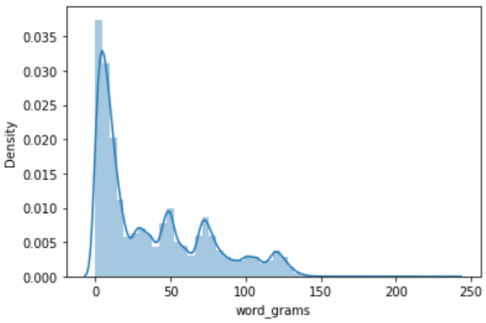



**Figure 3.18:** Word N-Grams distribution of DGA domain names

ones: this happens because, in general, DGA domains have low values of the Alexa N-Gram Matches, but the ones that have high values are very long strings, so this ratio will consequently be low. This feature helps in the distinction of the N-Gram matches values thanks to the fact that it takes into account also the SLM.

Obviously, this kind of analysis makes sense only with features that have continuous values. It would be useless with the features that can assume only 0 and 1 values.

All these features help the classifier to better distinguish between real and DGA-based domain names, but they are still not sufficient in order to have very

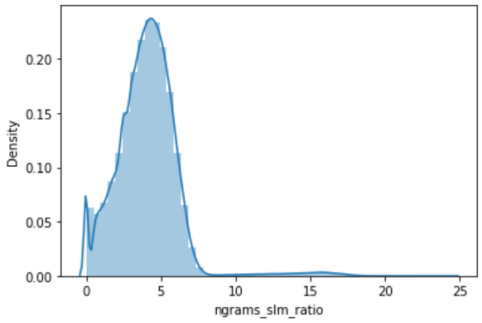



**Figure 3.19:** Alexa N-Grams/SLM Ratio distribution of legit domain names

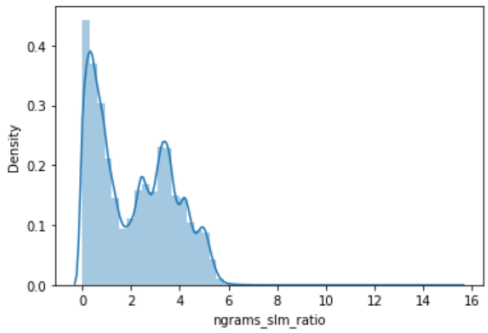



**Figure 3.20:** Alexa N-Grams/SLM Ratio distribution of DGA domain names

high performances because there are some families of Domain Generation Algorithm that create domains similar to the real ones.

After a phase of study of the conformation of the domain names, one of the things that most catches the eye is that real domains are mainly composed by dictionary words, mostly English words, while DGA-based ones are often made by a random sequence of characters, except for word-list-based Domain Generation Algorithms, as explained in chapter 1.

This analysis has led to the generation of another two features that analyze the composition of the domain names in terms of words. The first thing that can be verified is if a string contains a word. So, following this line of analysis, starting from a dictionary of English words, we compare the domain name with the dictionary, in order to check if it contains an English word.

The second feature, that is the most important for the classifiers in terms of distinction between legit and DGA-based domain names, as show by the Feature Importance in figure 3.21, is the word breaker.

The logic behind this feature is very simple: break the domain name and compare it with a dictionary of English words to check if it is composed exactly by a sequence of them. This word breaking is done because the major part of real domain names is a composition of words, so the classifier will be able to correctly label the domains

as valid or not. Even if there are word-list-based Domain Generation Algorithms, that generate domain names as a concatenation of words and can trick the classifier misleading it, the combination of these two features with the others will help it to better classify them.

At the end, the last, but not least, features generated for each domain are:

- starts_with_word, representing a flag that tells if the domain name starts with an English word (1) or not (0).

- word_breaker, representing a flag that tells if the domain name is composed by a sequence of English words (1) or not (0).

All the values, collected for each domain, are arranged within a Pandas dataframe. Figure 3.24 summarize the set of extracted features with a sample of domains.

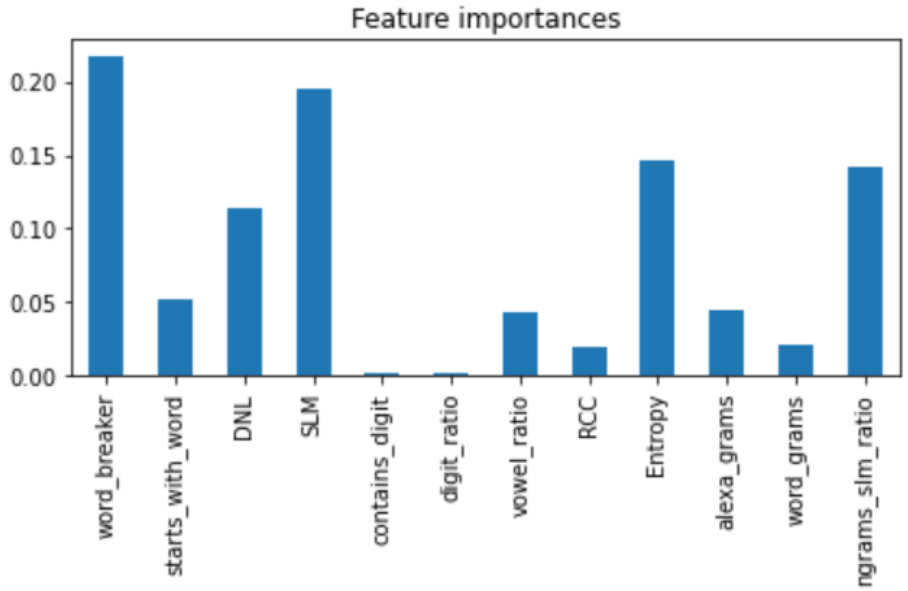



**Figure 3.21:** Feature Importance for the classifiers to label the domain names

## 3.4 DGA Detection: recognition of DGA-based domains

For the detection of DGA-based domain names, two different classification algorithms are used on the constructed dataset, which are then evaluated and compared, and the best of these will be chosen for continuation of the experiments.

The two approaches used are:

- Random Forest Classifier;
- XG-Boost Classifier.

### 3.4.1 Random Forest Classifier

The first classification algorithm used in the analysis is a basic Random Forest Classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting, as described in Figure 3.22.

Random forest is an ensemble learning method for classification, regression and other tasks that operates by constructing a set, so called "forest", of decision trees at training time. For classification tasks, so the task needed in this thesis work, the output of the random forest is the class selected by the majority of the trees.

Decision trees create a model that predicts the label of an item by evaluating a tree of if-then-else true/false feature questions, and estimating the minimum number of questions needed to estimate the probability of taking the correct decision. Decision trees can be mainly used for classification problems to predict a category, that is the goal of the DGA detection problem.

Random Forest Classifiers are better than Decision Trees in terms of over-fitting. The over-fitting problem is defined as the production of an analysis that corresponds too closely or exactly to the training set of data, and so may fail to fit

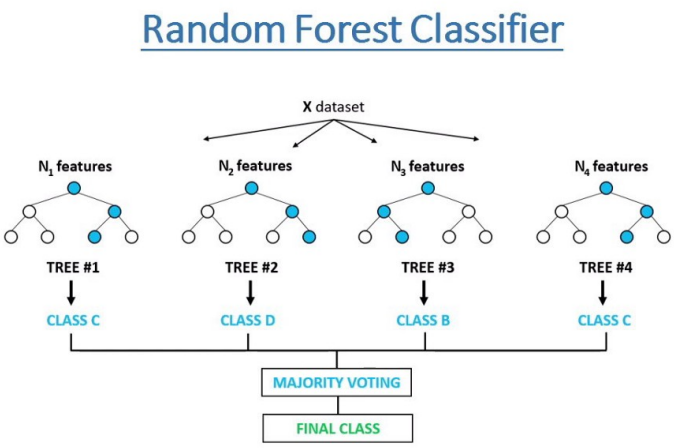



**Figure 3.22:** Random Forest Classifier graph example

to additional data or predict future observations in a reliable way. Random forests generally outperform decision trees. However, data characteristics can affect their performance.

Thanks to the scikit-learn library in Python, is very simple to create an instance of a Random Forest Classifier and perform some parameter tuning.

Parameter tuning consists of finding a set of optimal parameter values for a learning algorithm while applying this optimized algorithm to any data set. This combination of parameters is chosen in order to maximize the model's performances, minimize a predefined loss function to produce better results with fewer errors as possible.

After this phase of parameter tuning, the optimal set of hyper-parameters of the Random Forest Classifier for this classification problem are:

- n_estimators, representing the number of trees in the forest, equals to 100.
- max_depth, representing the maximum depth of the tree, equals to 5.
- criterion, representing the function to measure the quality of a split, equals to "gini". The Gini Impurity of a dataset is a number between 0-0.5, which

indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset.

- random_state, controlling both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node, equals to 1.

| Hyperparameter | Value |
|---|---|
| n_estimators | 100 |
| max_depth | 64 |
| criterion | gini |
| random_state | 1 |

**Table 3.1:** Hyperparameters used for the Random Forest Classifier

Once the parameter tuning phase is completed, the next step is training the classifier. To do so, the dataset is divided in one part (the 0.9 of the total dimension) used as training set, and another part (the 0.1 of the dataset) set aside as the hold out set, used for testing the classifier.

Within the training and testing phase, the Random Forest Classifier will predict every domain name in the hold out set as legit or DGA-based. The results obtained will be then discussed and evaluated in Chapter 5.

### 3.4.2 XG-Boost Classifier

The second classification algorithm used in the analysis is the XG-Boost Classifier.

XG-Boost stands for Extreme Gradient Boosting and it is a Decision-Tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. It provides parallel tree boosting and is one of the best machine learning libraries that can be applied to classification, regression and ranking problems, so it is very helpful in this thesis work for the classification of domain names. XG-Boost in built upon some concepts and algorithms: supervised machine learning, decision trees, ensemble learning and gradient boosting.

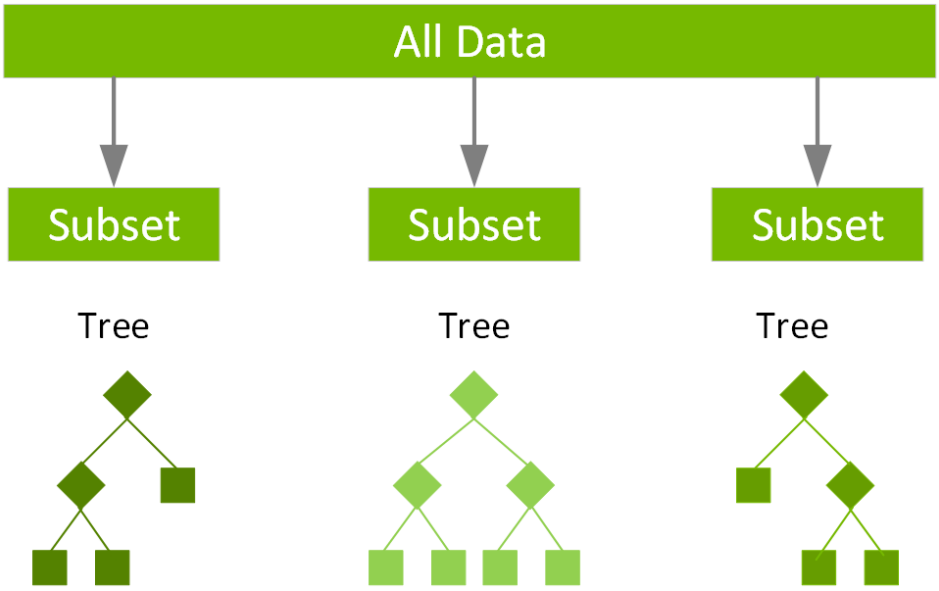



**Figure 3.23:** XG-Boost Classifier graph example

Like Random Forest Classifiers, XG-Boost ones are based on supervised machine learning. Supervised machine learning uses algorithms to train a model in order to find patterns in a dataset with features and labels, then it uses this trained model to predict the labels based on the features of a new dataset.

The Decision Trees have been already explained in section 3.4.1.

A so called Gradient Boosting Decision Trees (GBDT) is a Decision Tree ensemble learning algorithm similar to a Random Forest Classifier, used for classification and regression problems. Ensemble learning algorithms are a combination of multiple machine learning algorithms to obtain a better model. Random Forest and GBDT are based on the same idea: to build a model that consists of multiple decision trees, but the difference is in how to build and combine the them.

Gradient Boosting is based on the idea of improving, so "boosting", a single weak model by combining it with other weak models in order to generate an overall strong model.

The logic of GBDTs is to iteratively train an ensemble of Decision Trees, in which each iteration uses the error residuals of the previous model to fit the next one. At the end, the final prediction is a weighted sum of all of the tree predictions. Random Forest Classifiers minimize the variance and over-fitting problem, while XG-boost Classifiers minimize the bias and under-fitting.

The XG-Boost Classifier is a scalable and highly accurate implementation of the Gradient Boosting operation, it is built mainly for energizing machine learning model performances and computational speed. In the XG-Boost Classifier, trees are built in parallel, instead of sequentially combine them, like GBDT does. It makes a scanning operation across gradient values and using these partial sums to evaluate the quality of splits at every possible split in the training set.

With the xgboost package in Python is very simple to create an instance of an XG-Boost Classifier and perform some parameter tuning, an operation explained in Section 3.4.1.

After this phase of parameter tuning, the optimal set of hyper-parameters of the XG-Boost Classifier for this classification problem are:

- n_estimators, representing the number of trees in the forest, equals to 300.
- max_depth, representing the maximum depth of the tree, equals to 3.
- random_state, controlling both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node, equals to 1.

| Hyperparameter | Value |
|---|---|
| n_estimators | 300 |
| max_depth | 3 |
| random_state | 1 |

**Table 3.2:** Hyperparameters used for the XG-Boost Classifier

Once the parameter tuning phase is completed, the next step is training the classifier. To do so, the dataset is divided in one part (the 0.9 of the total dimension)

used as training set, and another part (the 0.1 of the dataset) set aside as the hold out set, used for testing the classifier.

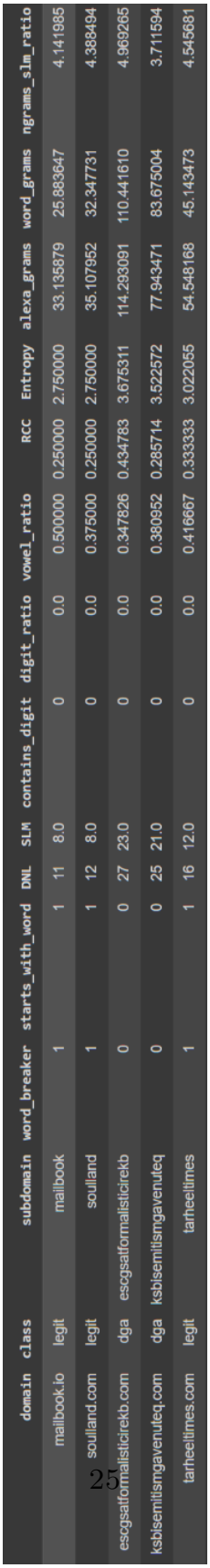
As said for Random Forest, within the training and testing phase, the XG-Boost Classifier will predict every domain name in the hold out set as legit or DGA-based. The results obtained will be then discussed and evaluated in Chapter 5.

## 3.5 Program language and support

For this approach, Python[9] is the programming language that is used. This language, in fact, is well suited for all those tasks of data analysis, data visualization and it is very simple make use of Machine Learning algorithms.

To support Python, the following libraries are used:

- Pandas[10], that is a library for data analysis;
- NumPy[11], a useful library for using mathematical functions;
- matplotlib[12], a library used for plots creation and visualization in Python;
- scikit-learn[13], a library that contains classification, regression, clustering algorithms and other machine learning tools in Python;
- publicsuffixlist[14], a library used for the verification of valid public suffix for a domain name;
- xgboost[15], a library used for the implementation of the XG-Boost Classifier.



| | domain | class | subdomain | word_breaker | starts_with_word | DNL | SLM | contains_digit | digit_ratio | vowel_ratio | RCC | Entropy | alexa_grams | word_grams | ngrams_slm_ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | mailbook.io | legit | mailbook | 1 | 1 | 11 | 8.0 | 0 | 0.0 | 0.500000 | 0.250000 | 2.750000 | 33.135879 | 25.883647 | 4.141985 |
| 1 | soulland.com | legit | soulland | 1 | 1 | 12 | 8.0 | 0 | 0.0 | 0.375000 | 0.250000 | 2.750000 | 35.107952 | 32.347731 | 4.388494 |
| 2 | escgsatformalisticirekb.com | dga | escgsatformalisticirekb | 0 | 0 | 27 | 23.0 | 0 | 0.0 | 0.347826 | 0.434783 | 3.675311 | 114.293091 | 110.441610 | 4.969265 |
| 3 | ksbisemitismgavenuteq.com | dga | ksbisemitismgavenuteq | 0 | 0 | 25 | 21.0 | 0 | 0.0 | 0.380952 | 0.285714 | 3.522572 | 77.943471 | 83.675004 | 3.711594 |
| 4 | tarheeltimes.com | legit | tarheeltimes | 1 | 1 | 16 | 12.0 | 0 | 0.0 | 0.416667 | 0.333333 | 3.022055 | 54.548168 | 45.143473 | 4.545681 |

**Figure 3.24:** Sample of domain names with the related extracted features

# Chapter 4

# Second Approach: Deep Learning models

Deep learning can be defined, in a simple way, as a subset of the machine learning, which is basically a neural network with three or more layers. The goal of these neural networks is to try to simulate the behavior of the human brain, making them able to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, the additional hidden layers can help to optimize the performances, especially in terms of accuracy.

The field of Deep learning is different with respect to the classical Machine Learning mainly for the type of data that it works with and the methods in which it learns.

Machine learning algorithms, like the ones seen in Chapter 3, are based on structured and labeled data to make predictions. So, starting from the input data, specific features are defined for the model and organized into tables. However, this does not necessarily mean that it does not use unstructured data, but it means that if it does, it generally goes through some pre-processing phase to organize them into a structured format.

Deep learning eliminates some of the data pre-processing operations that is involved in traditional Machine Learning. These algorithms can ingest and process directly unstructured data, like text and images, and it automates feature extraction, operation that needs human support in Machine Learning.

Then, through the processes of gradient descent and backpropagation, the deep learning algorithm adjusts and fits itself in terms of accuracy, allowing it to make predictions on new data with increased precision.

1. Gradient descent, in brief, is an optimization algorithm which is commonly used to train machine learning models and neural networks. The training set helps these models learn over time, while the cost function within the gradient descent acts as a barometer, measuring its accuracy with each iteration of parameter updates.

2. Back-propagation is the process based on the tuning of a neural network's weights to improve the prediction accuracy. It uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights of the function by moving backwards through the layers of the model helping to train it.

In the second approach of this experiment, the analysis of the domain names for the DGA detection problem is made directly on the string itself thanks to the application of some kind of deep learning models that can be used for the task of classification of them as legit or DGA-based.

## 4.1 Input Data

The dataset used for the Deep Learning approach is discussed in Section 3.1 and already used in the traditional Machine Learning method.

## 4.2 DGA Detection with Deep Learning

The second method used for the detection of DGA-based domain names, is based on Deep Learning models. These models are applied directly on the discussed dataset, using the technique known as text classification.

Text Classification is, in Artificial Intelligence (AI), an activity that has the purpose of classify documents, paragraphs, sentences or words, expressed in a natural language, assigning them, in an automatic way, one of the predefined labels. So, by using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign it one of the predefined categories based on its content.

This is the reason that led to take this type of analysis. With these models used for text classification, the domain names are analyzed as words, so that the model can learn the structure or some patterns that can help in the distinction between real and DGA-based domain names.

Two different Deep Learning models are used on the discussed dataset for the classification of domain names in legit or DGA-based. These models will be then evaluated and compared, also with the algorithms described in Section 3.4, and a final discussion will then be made.

The two models used are:

- Long Short-Term Memory (LSTM);
- Bidirectional LSTM.

### 4.2.1 Pre-Processing Data

Before starting with the definition and the implementation of the deep learning models, since this analysis is made on raw text (domain names), there is the need to pre-process the data. For text classification, this is done with the operation called Tokenization.

In order to make the model understand any type of text, there is the need to break the words down in a way that the model can understand them. That's why there is the introduction of the concept of tokenization in Natural Language Processing (NLP) problems. In other words, models can't work with text data if there is not the tokenization before.

Tokenization is basically a process that takes raw data as input and converts it into useful data strings. It is used in Natural Language Processing in order to split paragraphs, sentences or words into smaller units so that is simpler to assign a meaning.

Tokenization is the starting point of a NLP process, converting text input data into understandable bits of data so that a model can work with them.

There are different methods to tokenize text into tokens, a few examples are: word tokenization, character tokenization, sub-word tokenization. For this thesis work, the tokenization is made at character level, because the model has to deal with simple strings.

Tokenization at character level addresses some of the problems of the word one, because in word tokenization is difficult to separate unknown words and its accuracy is based on the dictionary it is trained with and it has to find a balance between accuracy and efficiency. Character tokenization does not have this dictionary problem, because the dictionary in this case is made of the characters of the language, but it has the drawback that the output size augments because there is the split in characters. It also adds an additional level of understanding the relationship between the characters of the strings.

Once the definition of the tokenization phase is finished, the next step is the definition and implementation of the deep learning model itself.

### 4.2.2 Long Short-Term Memory

The first Deep Learning model used in the analysis is an LSTM.

LSTM stands for Long Short-Term Memory. LSTM is a type of Recurrent Neural Network (RNN), but it has better performances with respect to traditional recurrent neural networks in terms of memory. LSTMs perform quite better thanks to the fact that they have a good hold over memorizing certain patterns. As with every other Neural Network, LSTM can have multiple hidden layers and, going through each layer, it keeps only the relevant information, while all the irrelevant information are discarded in every single cell. The structure of a LSTM is shown in Figure 4.1.

A hidden layer is placed between the input layer and the output one. It takes a set of weighted inputs and then produces an output using an activation function. This layer is called hidden because it is not part of the input or the output layer. This is the layer where all the processing happens.

Long Short-Term Memory has 3 main gates:

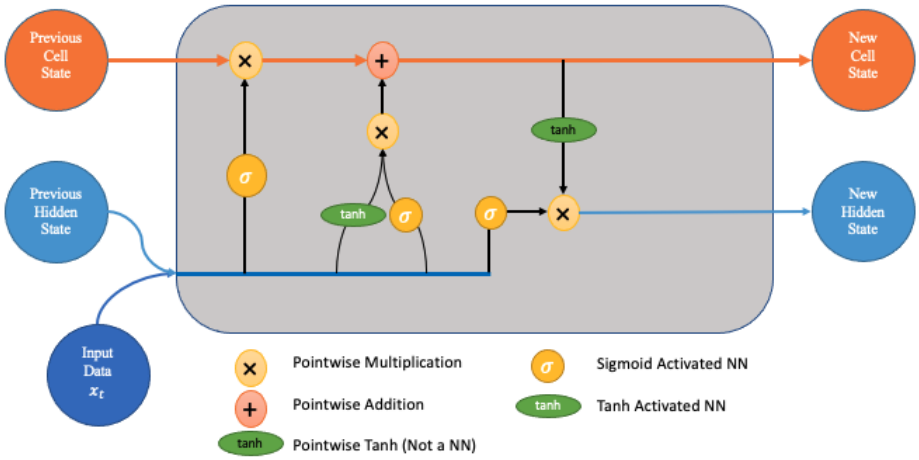



**Figure 4.1:** Long Short-Term Memory graph example

1. Forget Gate;
2. Input Gate;
3. Output Gate.

The Forget Gate has the responsibility of deciding which information has to be kept for calculating the cell state and which is not relevant and so can be discarded. Two inputs are given to the Forget Gate: the information from the previous hidden state, the previous cell, and the information from the current cell. They are passed through a sigmoid function and the ones that tend to 0 are discarded, while the others are passed ahead to calculate the cell state.

The Input Gate has the goal to update the cell state and decide which is the important information and which is not. As the Forget Gate helps to discard irrelevant information, the Input Gate helps to find out important information and store in memory data that are relevant.

The Output Gate has to decide what the next hidden state should be. The inputs are passed to a sigmoid function, then the updated cell state is passed through a tanh function and is multiplied with the output of the sigmoid to decide the information that the hidden state should carry.

One good reason to use LSTM for text classification is that it is effective in memorizing important information.

The first step is to define the model that will be used for the DGA detection problem. For this thesis work, the model chosen is a Sequential one. The Sequential model is a linear stack of layers, and so many layers can be added to it.

The first layer added to the Sequential model is the Embedding layer. It is commonly defined as the first hidden layer of a network and it is mainly used in Natural Language Processing (NLP) related problems, like the DGA Detection one. While dealing with textual data, they need to be converted into numbers before feeding into any Machine Learning model, also Neural Networks. The Embedding layer enables the conversion of the words into a fixed length vector of defined size.

After the Embedding layer, there is the Dropout one. The Dropout layer is a simple way to avoid the over-fitting problem in Neural Networks thanks to the fact that the outputs of a layer under Dropout are randomly sub-sampled. The Dropout layer, in other words, is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all the others. A Dropout layer can be applied to the input vector, in order to nullify some of its features, or to a hidden layer, in order to nullify some hidden neurons.

The next layer is a Long Short-Term Memory layer with 128 neurons, so dimensionality of the output space, that is the core of the model.

Then, another Dropout layer is added.

Finally, the last layer is the Dense one, that has 2 cells representing the 2 categories of the data, legit or DGA-based domain names. A Dense layer is a layer deeply connected with its preceding one, because each neuron of it receives inputs from all the neurons of previous layer. The other parameter passed to the Dense layer, besides the number of cells, is the Activation function.

An activation function is the function in an artificial neuron that, starting from

its inputs, delivers an output. Activation functions are very important for the role of the artificial neurons in Neural Networks. The activation function specified for the Dense layer will be the one to use. In Table 4.1, there are some examples of activation functions, the one chosen for this experiment is softmax.

| | |
|---|---|
| ReLU | $f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$ |
| Softmax | $f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} i = 1, ..., J$ |
| tanh | $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ |

**Table 4.1:** Examples of activation functions

After having defined all the layers of the Sequential model, there is the compilation phase, in which there is the definition of the optimizer, the loss function and the metrics used.

The optimizer is a function or an algorithm that has the task of modifying the attributes of the Neural Network like weights and the learning rate. it helps in reducing the overall loss and improve the accuracy, so its purpose is to adjust model weights to maximize a loss function.

A loss function compares the expected and the predicted output values and then measures how good is the Neural Network in modelling the training data. So, it is used as a way to measure how well the model is performing. In Table 4.2, there are some activation functions. The one used for this model is the Binary Cross-Entropy, because the DGA Detection problem is a binary classification one.

A metric can be defined simply as a function used to judge the performance of a model. They are similar to loss functions, except that the results from evaluating a metric are not used when training the model.

As for the classical Machine Learning algorithms, the dataset is divided in one

| | |
|---|---|
| MSE / L2 Loss / Quadratic Loss | $\frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{N}$ |
| (Binary) Cross Entropy<br>(average reduction on higher dimensions) | $\frac{\sum_{i=1}^{N} \sum_{j=1}^{C} \hat{y}_i \log (y_{i,j})}{N}$ |
| Categorical Cross Entropy<br>(sum reduction on higher dimensions) | $-\sum_{i=1}^{N} \hat{y}_i + \log \left( \sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \right)$ |

**Table 4.2:** Examples of loss functions

part (the 0.9 of the total dimension) used as training set, and another part (the 0.1 of the dataset) set aside as the hold out set, used for testing the model itself.

With the keras package in Python is very simple to create an instance of the explained model with all its features. After having defined and compiled the model, the next step is to train it and fit with the training data and the subsequent set of hyper-parameters:

- epochs, representing the number of iterations over the training data, equals to 5;
- batch_size, representing the number of samples per gradient update, equals to 64;
- validation_split, representing the part of the training data used as validation data, equals to 0.11;
- the callback, that is an object that can perform actions at various stages of training phase, used is Early Stopping.

| Hyperparameter | Value |
|---|---|
| epochs | 5 |
| batch_size | 64 |
| validation_split | 0.11 |
| callback | Early Stopping |

**Table 4.3:** Hyperparameters used for the LSTM

The Early Stopping callback is chosen because it can stop training when the monitored metrics, validation loss in this case, has stopped improving.

Within the training and testing phase, the Long Short-Term Memory (LSTM) model will predict every domain name in the hold out set as legit or DGA-based. The results obtained will be then discussed and evaluated in Chapter 5.

### 4.2.3 Bidirectional Long Short-Term Memory

The second Deep Learning model used in this thesis work is a Bidirectional LSTM.

Bidirectional Long Short-Term Memory (BiLSTM) is the process that makes any neural network to have the sequence information in both directions, backwards or forward. It is a kind of Recurrent Neural Network (RNN) used mainly for Natural Language Processing problems and is also a powerful tool for modeling the sequential dependencies between sentences and words in both the directions of the sequence.

In bidirectional, the input data flows in two directions, making a BiLSTM different from a regular LSTM. This is because in the regular LSTM the input data can flow in one direction, either backwards or forward, while in bidirectional the input flows in both directions, as shown in Figure 4.2, to preserve the future and the past information.

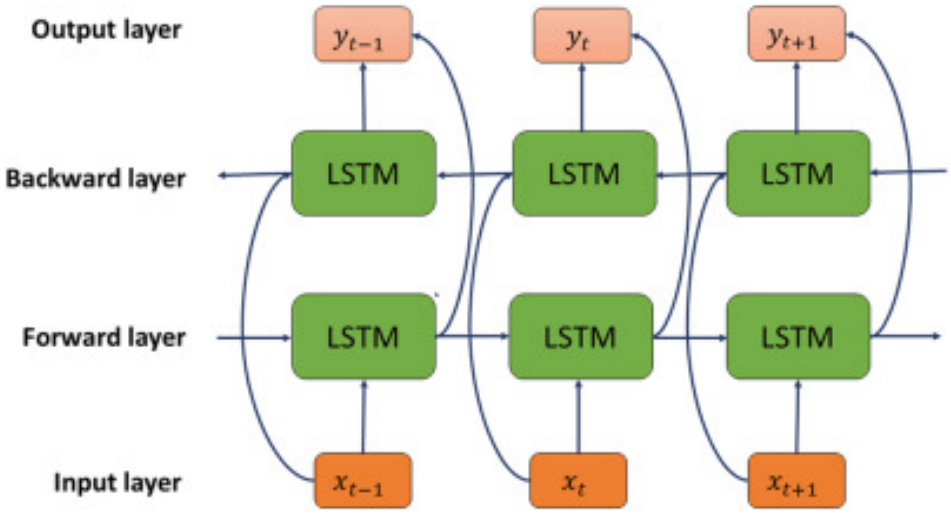



**Figure 4.2:** Bidirectional Long Short-Term Memory graph example

In other words, BiLSTM adds one more LSTM layer to the model, which reverses the direction of information flow. This means that the input sequence flows backward in the LSTM layer added to the model. Finally, the outputs from both LSTM layers can be combined in several ways, for example averaging, summing, multiplying or concatenating them.

In many real-word problems, especially in Natural Language Processing, this type of architecture has many advantages. One of the reasons is that in an input sequence, that can be a word or a sentence, every component has information in both directions. To address this problem, BiLSTM has the ability to produce a more meaningful output thanks to the combination of the LSTM layers from both directions.

BiLSTM has also some disadvantages: in fact, it is a much slower model with respect to regular LSTM and requires more time for the training phase. The Bidirectional Long Short-Term Memory model is very useful in some NLP tasks, such as sentence classification, and this goes in the direction of addressing the DGA Detection problem.

Also in this method, the first step is to define the model that will be used for the DGA detection problem. The model chosen is a Sequential one, the same as for the regular LSTM model. Starting from this point, there is the definition of the sequential layers of the model.

The structure of the Sequential model is equals to the one chosen for the regular Long Short-Term Memory model.

The first layer of the model is the Embedding one, also known as the first hidden layer of a Neural Network. After that, there is the first Dropout layer, with rate equals to 0.3, that is the fraction of the input units to drop at each step during training time, same rate as for the previous model. The only difference between the two models is the replacement of the regular LSTM with the BiLSTM, with the same number of neurons. Then, there is another Dropout layer and finally the model ends with the Dense layer.

All the details of the layers included in the model are explained in the Section 4.2.2 because, as just said, the composition is the same, except for the core of the model.

As for all the previously described approaches, also here the dataset is divided in one part (the 0.9 of the total dimension) used as training set, and another part (the 0.1 of the dataset) set aside as the hold out set, used for testing the model itself.

After having defined and compiled the model, the next step is to train it and fit with the training data and the subsequent set of hyper-parameters:

- epochs, representing the number of iterations over the training data, equals to 5;
- batch_size, representing the number of samples per gradient update, equals to 64;
- validation_split, representing the part of the training data used as validation data, equals to 0.11;
- the callback, that is an object that can perform actions at various stages of training phase, used is Early Stopping.

Also the Early Stopping callback is explained in Section 4.2.2.

| Hyperparameter | Value |
|---|---|
| epochs | 5 |
| batch_size | 64 |
| validation_split | 0.11 |
| callback | Early Stopping |

**Table 4.4:** Hyperparameters used for the BiLSTM

Within the training and testing phase, the Bidirectional Long Short-Term Memory (BiLSTM) model will predict every domain name in the hold out set as legit or DGA-based. The results obtained will be then discussed and evaluated in Chapter 5.

## 4.3 Program language and support

For this approach, Python[9] is the programming language that is used. This language, in fact, is well suited for all those tasks of data analysis, data visualization and it is very simple make use of Machine Learning algorithms.

To support Python, the following libraries are used:

- Pandas[10], that is a library for data analysis;
- NumPy[11], a useful library for using mathematical functions;
- matplotlib[12], a library used for plots creation and visualization in Python;
- scikit-learn[13], a library that contains classification, regression, clustering algorithms and other machine learning tools in Python;
- publicsuffixlist[14], a library used for the verification of valid public suffix for a domain name;
- keras[16], a library used for defining, compiling, training and testing the Deep Learning models.

# Chapter 5

# Metrics and Results

Because the experiments studied in this thesis work were, for both the approaches used, basically a classification problem, analyzing domain names and labelling them as legit or generated by a Domain Generation Algorithm, the following metrics for evaluating the results are taken into account.

Given:

1. TP as the number of True Positives, i.e. the number of domain names correctly classified as malicious;

2. TN as the number of True Negatives, i.e. the number of domain names correctly classified as legit;

3. FP as the number of False Positives, i.e. the number of domain names incorrectly classified as malicious;

4. FN as the number of False Negatives, i.e. the number of domain names incorrectly classified as legit;

the Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

The Accuracy represents the ratio of the number of correct predictions to the total number of observations.

The Precision and the Recall, instead, are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

and

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

The Precision is the ratio of the number of correct positives predictions (DGA-based domain names) to the total number of observations predicted as positives, while the Recall is the ration of the number of correct positives predictions to the total number of actually positives.

Finally, the F1-Score is defined as:

$$F1 - Score = 2 * \frac{P * R}{P + R} \tag{5.4}$$

where P is the Precision and R is the Recall. The F1-Score takes into account both of the previous metrics, in fact it represents their harmonic mean. In this thesis work, it is one of the metrics taken into account the most.

All these metrics are analyzed through a Classification Report. The reason why the Classification Report is used, is to measure the quality of the predictions of a classification algorithm. How many predictions are True and how many are False.

More specifically, the computation of True Positives, False Positives, True negatives and False Negatives are used to predict the metrics, as explained before. So, precision, recall and f1-score are shown in this report on a per-class basis. In the classification report there are also the averages. These averages include:

- macro average, representing the average of the unweighted mean per label;
- weighted average, representing the average of the support-weighted mean per label;
- sample average (only for multi-label classification).

The other kind of report used to analyze the results is the Confusion Matrix. A confusion matrix can be defined as a technique useful to summarize the performance

of a classification algorithm. This matrix is very helpful because, for performance analysis, the classification accuracy alone is misleading and hides the details needed to better understand the performance of a classification model. The structure of a Confusion Matrix is explained in Figure 5.1.

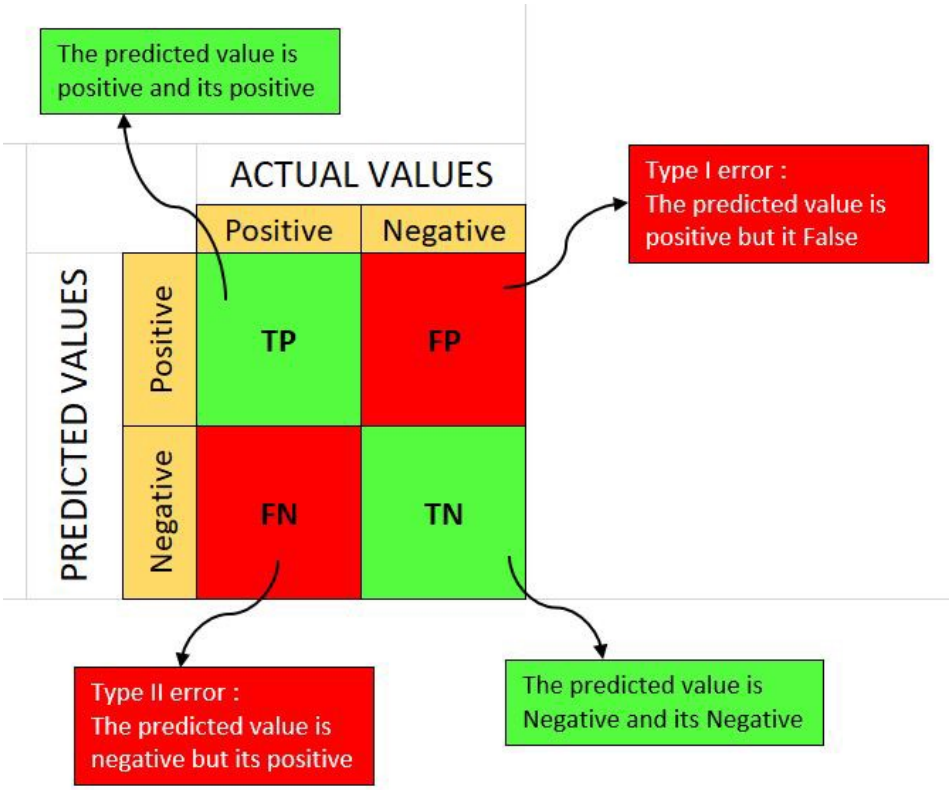



**Figure 5.1:** Example of a Confusion Matrix

In a confusion matrix there is the summary of the number of correct and incorrect predictions with count values and they are divided per class. The confusion matrix takes the expected outcomes and the predictions, and outputs

1. the number of correct predictions for each class;
2. the number of incorrect predictions for each class, organized by the class that was predicted.

These numbers are finally organized in the matrix and ready for the analysis. Basically, it shows the number of True Positives, False Positives, True Negatives and False Negatives.

Now that the metrics that are the basis of the validation and the reports used to show them have been described, the next step is to analyze the results of every method used, and compare them utilizing the metrics explained above (thanks to the classification report), but also doing an analysis on the individual TP, TN, FP and FN numbers (done with the confusion matrix).

## 5.1 Validation Method

The validation of the algorithms and models used in this thesis work is made with the train test split.

The objective of supervised learning is to create a model that performs well on new data, but if there are not new data, there is the possibility to simulate this situation with a procedure called train test split, that is a model validation process that allows the simulation of how the model would perform with unseen data.

First, the dataset is divided in two parts: the training and the testing set. This is done with a random sampling without replacement of a previously decided percentage of the entire dataset. Then, the model is trained on the training set and, finally, it is evaluated on the testing set.

This kind of validation of the algorithms and models used is made on two different case of study.

The first one is the basic division of the dataset in training and testing set. In this case, from the two dataframes of Alexa domain names and Domain Generation Algorithm-based domain names, the 90% of the domains are taken as training set, while the 10% left is put aside as hold out set that will be used for the validation part as testing set. Both the dataframes are randomly splitted and then merged: training legit domains with training DGA-based ones and testing legit domains with testing DGA-based ones.

In the second analysis, there is a finer work made on the DGA-based domains. Since that the common case is that the algorithm or model will encounter new domain names, probably generated from a new family of Domain Generation

Algorithm, the idea is to simulate this situation, making the algorithm or model encounter something it has never seen, that is, domains from new families.

For this purpose, the DGA-based domain names dataframe is again divided in 90% of training and 10% of testing set, but this time this division is not random, but there is the selection of a few number of DGA families taken as testing set, obviously making sure that the size is always the 10% of the dataset, and the remaining 90% of families as training set.

The Alexa domain names, instead, are again divided randomly and the two sets are merged as in the first case.

## 5.2 Results of traditional Machine Learning Methods

The first analysis of results is done on the two approaches that make use of structured data, such as the Random Forest Classifier and the XG-Boost Classifier.

### 5.2.1 Random Forest Classifier

The results of the first validation of the Random Forest Classifier are shown in Figure 5.2 for the classification report and in Figure 5.3 for the confusion matrix.

The overall Accuracy of the Random Forest for this validation case is about 93%, but this metric alone is not very useful and exhaustive.

As shown in the classification report in Figure 5.2, the values of the Precision for the two classes are quite similar to the overall Accuracy, meaning that there is a few number of False Positives. Also the Recall has quite high values, so the number of False Negatives is low, especially for legit domain names. Finally, the F1-Score has values that are comparable with Precision and Recall.

Taking into account directly the numbers of TP, FP, TN and FN, as shown in Figure 5.3, the Random Forest Classifier performs well in labelling the legit domain

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| legit | 0.92 | 0.94 | 0.93 | 93089 |
| dga | 0.94 | 0.93 | 0.93 | 98912 |
| accuracy | | | 0.93 | 192001 |
| macro avg | 0.93 | 0.93 | 0.93 | 192001 |
| weighted avg | 0.93 | 0.93 | 0.93 | 192001 |

**Figure 5.2:** Classification report of the first validation of the Random Forest Classifier

names, with the percentage of True Negatives very high (almost 94%) and a few number of False Positives (around 6%).

This is a good thing because the algorithm does not have to signal too many alarms, but only the real anomalous values. In fact, the classifier is quite good in detecting real DGA-based domain names (almost 93% of True Positives), while it labels more DGA-based domains as valid (7% of False Negatives) with respect to valid domains labelled as DGA-based.

From these metrics and percentages, it turns out that the Random Forest Classifier is performing quite well in detecting real anomalies.

The results of the second validation of the Random Forest Classifier, so with distinct families of DGA as testing set, are shown in Figure 5.4 for the classification report and in Figure 5.5 for the confusion matrix.

The first difference to point out is that the overall Accuracy in this case drops, from the 93% of the first simulation, to about 88%. Now that this metric values as been compared, a deeper analysis can give a better and exhaustive explanation.

As shown in the classification report in Figure 5.4, the values of the Precision for the two classes are different, 83% for legit domains and 94% for DGA-based ones, meaning that there is a quite high number of DGA-based domains labelled

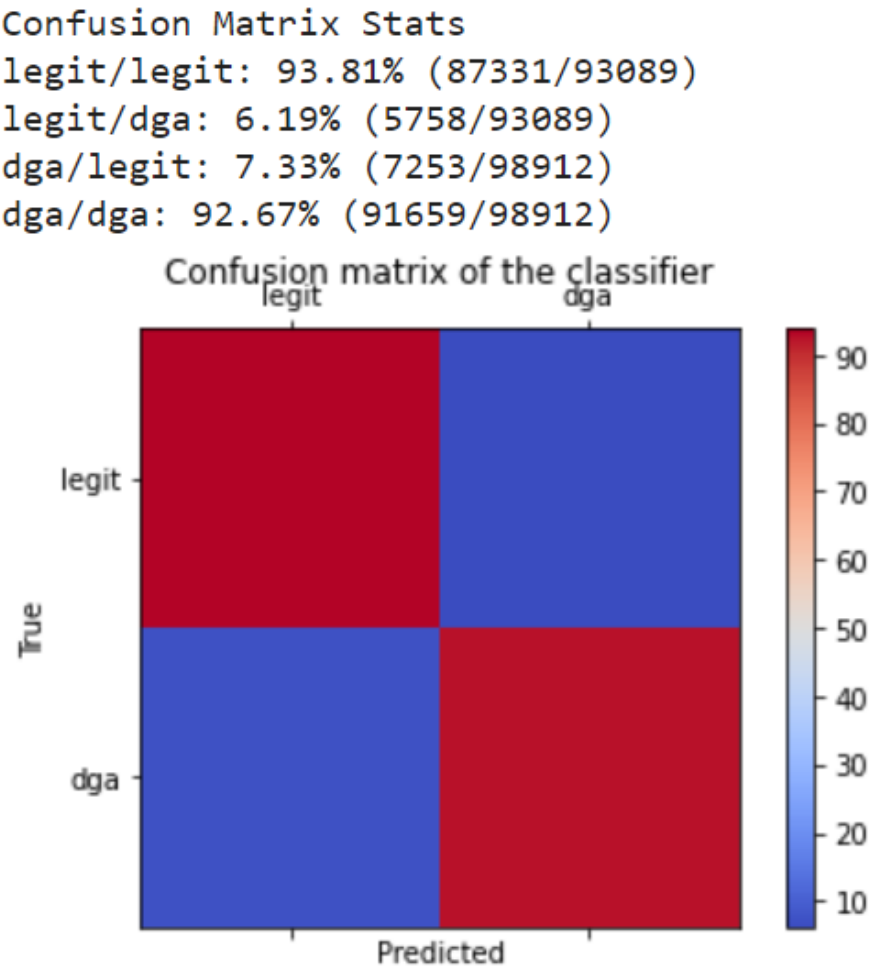



**Figure 5.3:** Confusion matrix of the first validation of the Random Forest Classifier

as legit, while the number of legit domains labelled as anomalous remains low.

Also the Recall has different values, so the number of False Negatives is quite high, as highlighted by the 82% for the DGA class. Finally, the F1-Score values drops too, because of the number of False Negatives.

Going deeper in the analysis of the results and talking about the numbers of TP, FP, TN and FN, shown in Figure 5.5, the first thing to highlight is that the Random Forest Classifier continues to perform well in labelling the legit domain names, as shown by the percentage of True Negatives (more than 94%) and a few number of False Positives (slightly less 6%). This remains a good news because the

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| legit | 0.83 | 0.94 | 0.89 | 93089 |
| dga | 0.94 | 0.82 | 0.88 | 98905 |
| accuracy | | | 0.88 | 191994 |
| macro avg | 0.89 | 0.88 | 0.88 | 191994 |
| weighted avg | 0.89 | 0.88 | 0.88 | 191994 |

**Figure 5.4:** Classification report of the second validation of the Random Forest Classifier

algorithm does not have to signal too many alarms, but only the real anomalous values.

The real problem is that the number of False Negatives, so DGA-based domains labelled as valid, has increased up to almost 18% and, consequentially, the number of True Positives has decreased, with the percentage fell to 82%.

This means that the Random Forest Classifier performs well when it has to label something that it knows, while, if it encounters something that it has never seen before, it has more difficulty. So, if the classifier encounters domains generated from new families with different characteristics with respect to the ones known to it, it's difficult for the Random Forest to detect them.

### 5.2.2 XG-Boost Classifier

For the XG-Boost Classifier, the results of the first validation are shown in Figure 5.6 for the classification report and in Figure 5.7 for the confusion matrix.

The overall Accuracy of the XG-Boost for this validation case is about 96%, but, as for the Random Forest, this metric alone is not very useful and exhaustive, even if, at a first look, this approach seems quite better.

As shown in the classification report in Figure 5.6, the values of the Precision for

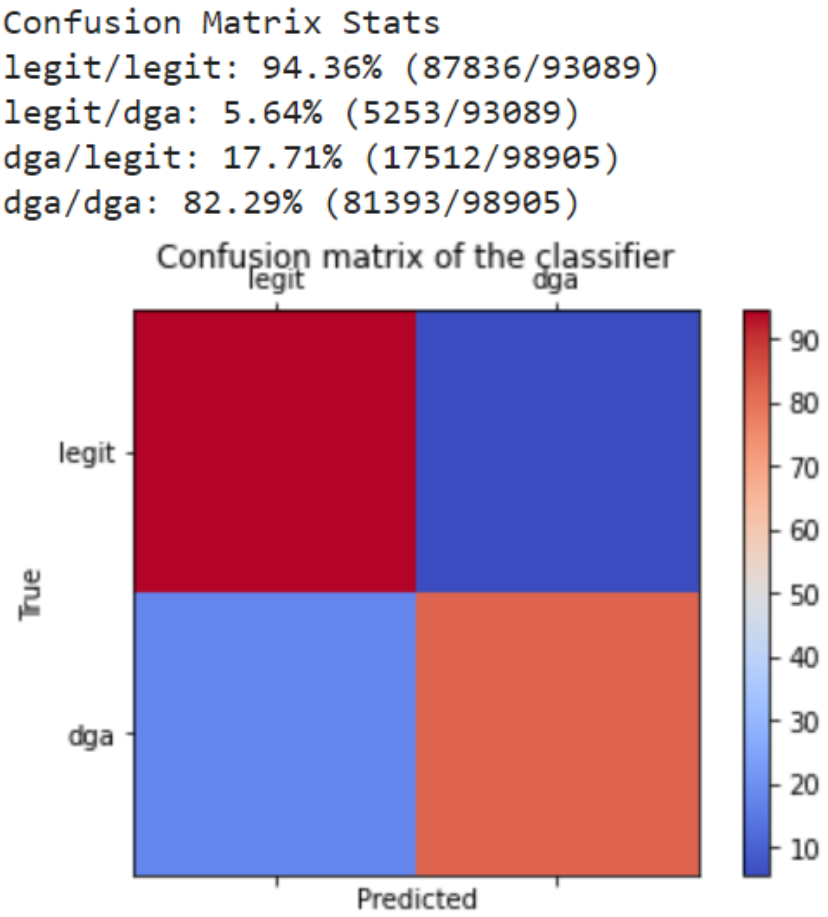



**Figure 5.5:** Confusion matrix of the second validation of the Random Forest Classifier

the two classes are identical to the overall Accuracy, meaning that the number of mis-classified domain names is quite low for both the two classes. Also the Recall has quite high values, so also the number of False Negatives is low for both classes. Finally, the F1-Score has the same values as the Precision and the Recall.

With the analysis of the percentages of TP, FP, TN and FN, shown in the confusion matrix in Figure 5.7, the first thing to notice is that the XG-Boost Classifier performs well in labelling correctly the legit domain names, with the percentage of True Negatives very high (around 96%), but also the DGA-based ones, with a percentage of True Positives around 96% too.

Accuracy: 96.07%

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| legit | 0.96 | 0.96 | 0.96 | 93089 |
| dga | 0.96 | 0.96 | 0.96 | 98912 |
| accuracy | | | 0.96 | 192001 |
| macro avg | 0.96 | 0.96 | 0.96 | 192001 |
| weighted avg | 0.96 | 0.96 | 0.96 | 192001 |

**Figure 5.6:** Classification report of the first validation of the XG-Boost Classifier

Consequently, the percentages of False Positives and False Negatives are similar (slightly less than 4%), with slightly less legit domain names labelled as anomalies with respect to the opposite situation. That is very good because the algorithm has to signal fewer number of alarms as possible, so trying to reduce as much as possible the number of False Positives.

From these metrics and from the percentages of TP, FP, TN, FN, it turns out not only that the XG-Boost Classifier performs very well in the DGA Detection problem, but also that it has better performances than the Random Forest Classifier.

The results of the second validation of the XG-Boost Classifier, with DGA families that the algorithm as never seen as testing set, are shown in Figure 5.8 for the classification report and in Figure 5.9 for the confusion matrix.

Also in this case, the first difference to point out is that the overall Accuracy drops, from the 96% of the first simulation, to about 92%, but still remaining a quite good value if compared to the one of the Random Forest Classifier.

The overall Accuracy is a starting point of the comparison, so going deeper in the analysis can give a better and exhaustive explanation of the behavior of the classifier.

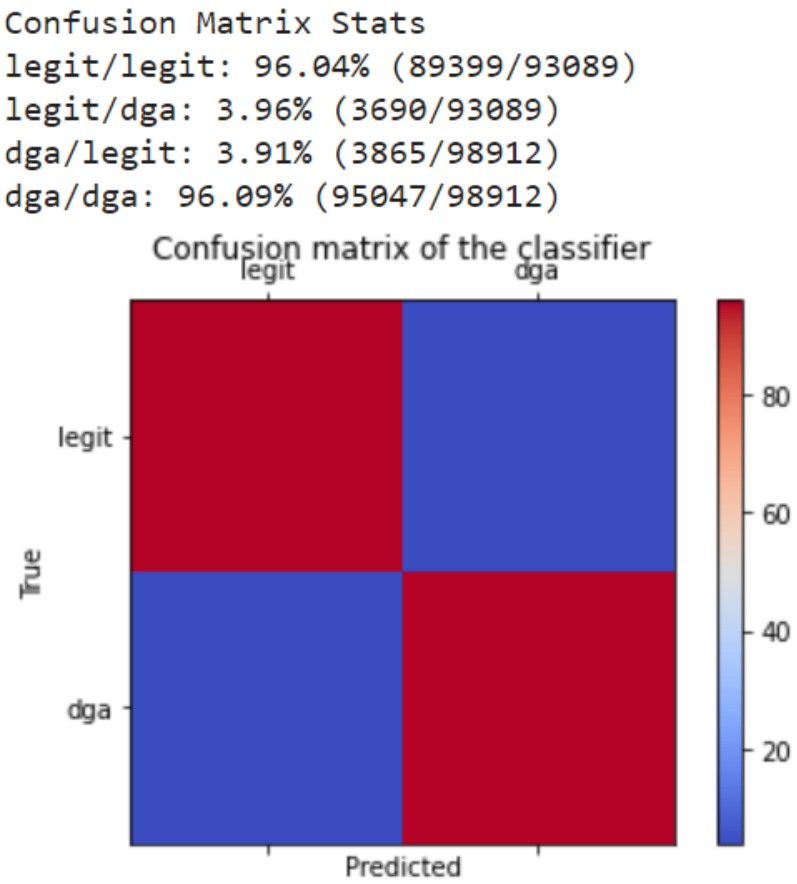



**Figure 5.7:** Confusion matrix of the first validation of the XG-Boost Classifier

As shown in the classification report in Figure 5.8, the values of the Precision for the two classes are different, 89% for legit domains and 96% for DGA-based ones, meaning that there is an higher number of DGA-based domains labelled as legit, while the number of legit domains labelled as anomalous remains very low, with numbers similar to the ones of the first simulation.

Also the Recall has different values, exactly the opposite of the ones of the Precision, so the number of False Negatives is quite high, as shown by the 89% for the DGA class. Finally, because of the number of False Negatives, the F1-Score values drops too, but with acceptable values.

Accuracy: 92.39%

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| legit | 0.89 | 0.96 | 0.92 | 93089 |
| dga | 0.96 | 0.89 | 0.92 | 98905 |
| accuracy | | | 0.92 | 191994 |
| macro avg | 0.93 | 0.93 | 0.92 | 191994 |
| weighted avg | 0.93 | 0.92 | 0.92 | 191994 |

**Figure 5.8:** Classification report of the second validation of the XG-Boost Classifier

Taking into account also the numbers of TP, FP, TN and FN, shown in the confusion matrix in Figure 5.9, the XB-Boost Classifier continues to perform well in labelling the legit domain names, so it is able to recognize in almost all cases which are the valid domains, as shown by the percentage of True Negatives (more than 96%) and a few number of False Positives (slightly less 4%). As said before, this is always a good news because the algorithm does not have to signal too many alarms, but only the real DGA-based domains.

The number of False Negatives, so DGA-based domains labelled as valid, is again a problem for the classifier because it has increased up to slightly more than 11% and, consequentially, the number of True Positives has decreased, with the percentage that has fell to around 89%.

This analysis highlights that the XG-Boost Classifier performs very well when it has to label something that it knows, while, if it encounters something that it has never seen before, like new Domain Generation Algorithm families with different characteristics with respect to the ones known to it, it has more difficulty in detecting them, even if the performances remains quite good.

So, as a first comparison, the XG-Boost Classifier performs better than the Random Forest Classifier in both the simulations. Obviously, both the classifiers decrease their performances in the second validation, but XG-Boost is much more robust in the situation in which it encounters new DGA families.

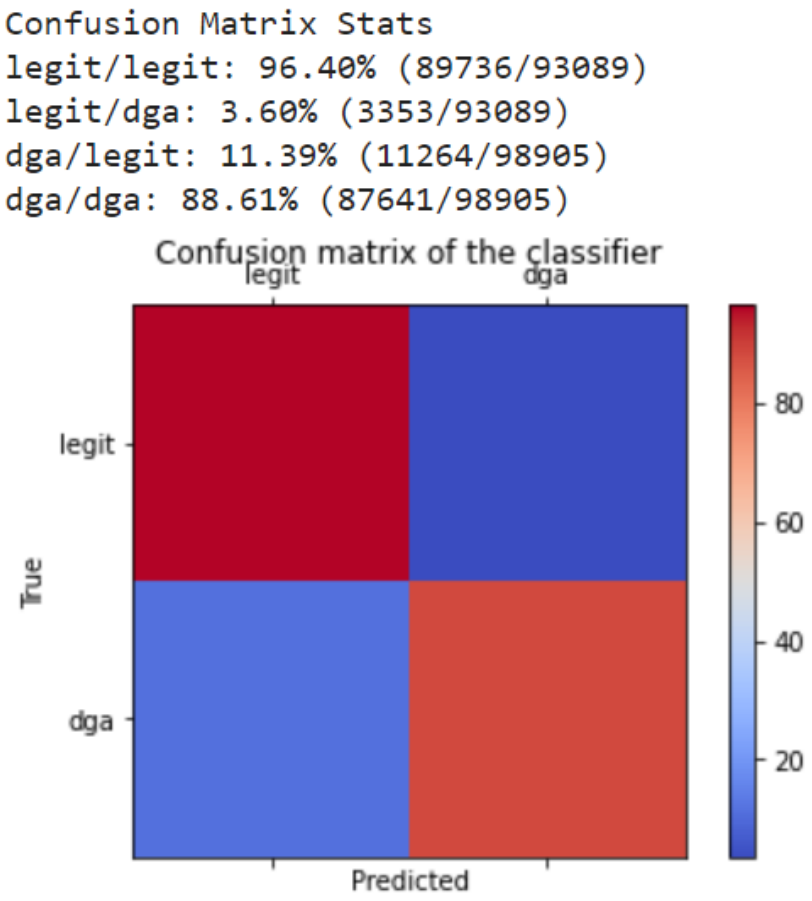



**Figure 5.9:** Confusion matrix of the second validation of the XG-Boost Classifier

## 5.3 Results of Deep Learning Methods

The second analysis of results is done on the two approaches that are based on text classification, such as the Long Short-Term Memory (LSTM) and the Bidirectional LSTM models.

### 5.3.1 Long Short-Term Memory Model

For the Long Short-Term Memory Model, the results of the first validation are shown in Figure 5.10 for the classification report and in Figure 5.11 for the confusion matrix.

The overall Accuracy of the LSTM Model for the first validation case is about 98.5%. So, at first glance, the accuracy tells that this model has very high performances and looks promising, but, as said for the other classifiers, this metric alone is not very useful and exhaustive.

As shown in the classification report in Figure 5.10, the values of the Precision for the two classes are similar to the overall Accuracy, meaning that the number of mis-classified domain names is quite low for both the two classes, with an accent on the Precision of the DGA class, that is higher than 99%, showing that the number of mis-classified valid domains is lower than the mis-classified DGA-based domains.

Also the Recall has quite high values for both classes, around 98%, meaning that also the number of False Negatives is very low. Finally, the F1-Score has values similar to the Precision and the Recall, between 98% and 99%.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.9827 | 0.9896 | 0.9861 | 93090 |
| 1 | 0.9902 | 0.9836 | 0.9869 | 98912 |
| accuracy | | | 0.9865 | 192002 |
| macro avg | 0.9864 | 0.9866 | 0.9865 | 192002 |
| weighted avg | 0.9865 | 0.9865 | 0.9865 | 192002 |

**Figure 5.10:** Classification report of the first validation of the Long Short-Term Memory Model

After having analyzed the metrics used, the study of the percentages of TP, FP, TN and FN, shown in the confusion matrix in Figure 5.11, can explain in a more detailed way the behavior of the Long Short-Term Memory Model.

The first thing that is highlighted is that the LSTM Model performs very well in labelling correctly the domain names, both the valid ones, with the percentage of True Negatives very high (slightly less than 99%), and the DGA-based ones, with the percentage of True Positives that is more than 98%.

Consequently, the percentages of False Positives and False Negatives are similar and very low (slightly more than 1%), with the number of legit domain names labelled as anomalies that is fewer with respect to the number of DGA-based ones labelled as valid. This is always a very good situation because the algorithm has to signal fewer number of alarms as possible, so the number of False Positives should be very low.

From the metrics analyzed and from the percentages of TP, FP, TN, FN, it turns out not only that the LSTM Model performs very well in the DGA Detection problem and the results are quite perfect, but also that it has better performances with respect to the traditional Machine Learning classifiers.

The results of the second validation of the Long Short-Term Memory Model, with a separated set of DGA families that the algorithm as never seen as testing set, are shown in Figure 5.12 for the classification report and in Figure 5.13 for the confusion matrix.

As for the previous classifiers, the first difference to point out in the two validations is that the overall Accuracy drops, from the 98% of the first simulation, to about 86%, that is a value comparable to the one obtained from the traditional Machine Learning classifiers.

The overall Accuracy can be a valid starting point for the comparison of the performances, so, with a more deeper analysis, the behavior of the model can be explained in a better and exhaustive way.

As shown in the classification report in Figure 5.12, the values of the Precision for the two classes are very different, 81% for legit domains and 93% for DGA-based ones. This means that there is a high number of DGA-based domain names that are labelled as legit, but the number of valid domains labelled as anomalous remains quite low, with numbers slightly less than the ones of the first simulation.

Also the Recall has different values for the two classes, the opposite of the ones of the Precision. This is because the number of False Negatives is quite high, as shown by the 79% for the DGA class, that is a very low value. Finally, because of

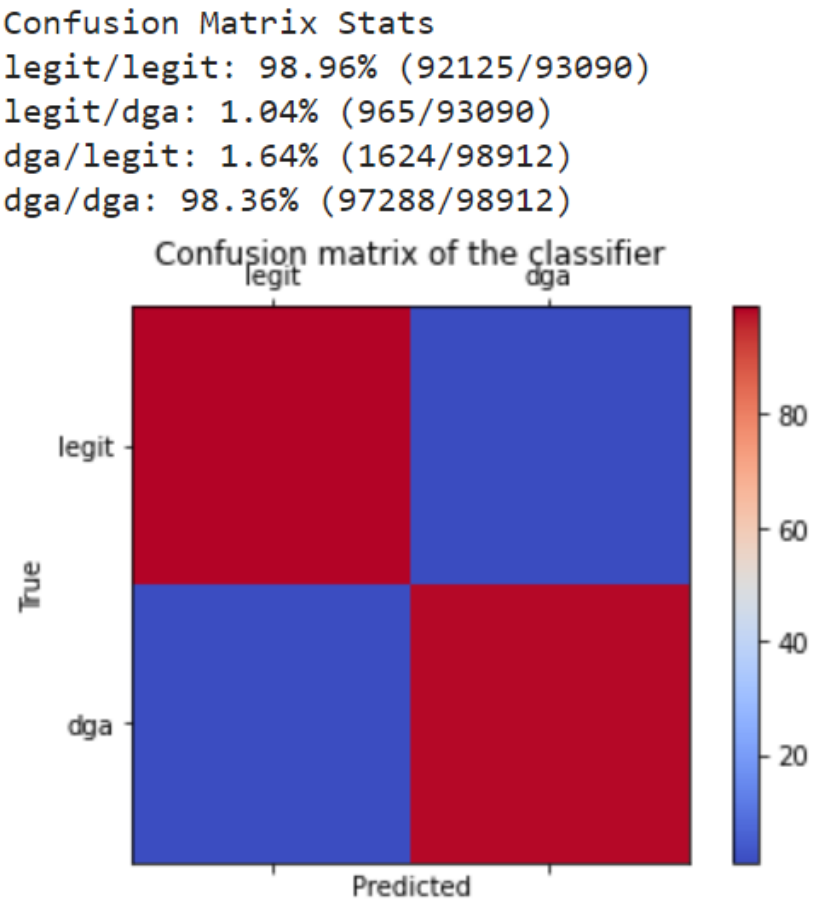



**Figure 5.11:** Confusion matrix of the first validation of the Long Short-Term Memory Model

the number of False Negatives of the two classes, the F1-Score values drops too, but with similar values.

In this second simulation, the analysis and study of the TP, FP, TN and FN, shown in the confusion matrix in Figure 5.13, are very helpful in describing the behavior of the Long Short-Term Memory Model in a detailed way.

The LSTM Model continues to perform well in labelling the legit domain names, even if with results not equals to the ones of the first simulation. In general, it is

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.8099 | 0.9396 | 0.8700 | 93090 |
| 1 | 0.9331 | 0.7924 | 0.8570 | 98905 |
| accuracy | | | 0.8638 | 191995 |
| macro avg | 0.8715 | 0.8660 | 0.8635 | 191995 |
| weighted avg | 0.8734 | 0.8638 | 0.8633 | 191995 |

**Figure 5.12:** Classification report of the second validation of the Long Short-Term Memory Model

able to recognize with optimal results which are the valid domain names, as shown by the percentage of True Negatives (slightly less than 94%) and a few number of False Positives (slightly more than 6%). As said for the previous validation, this is a good news because the algorithm has to signal the fewer number of alarms as possible, trying to alert only in presence of real DGA-based domains.

The number of False Negatives, so DGA-based domains labelled as valid, is the real problem of this simulation for the LSTM Model, because it has increased up to around 20% and, consequentially, the number of True Positives has decreased, with the percentage that has fell to around 79%.

This analysis highlights that the LSTM Model has performances that are quite perfect when it has to label something that it has already seen. But, if it encounters something that it has never seen before, like the situation with the presence of new Domain Generation Algorithm families with different characteristics with respect to the ones known to it, it has more difficulty in detecting them.

This problem can be explained with the fact that the Long Short-Term Memory Model is over-fitting, so it learns very well from the training set and it is quite perfect in classifying something similar to the things present in the training set, but if it encounters something new, it has difficulties because it is too much based on what it has learned.

So, comparing the LSTM Model with the two traditional Machine Learning Classifiers, it has better performances in the first simulation. For the second one, it has performances similar to the ones of the Random Forest Classifier, but worst

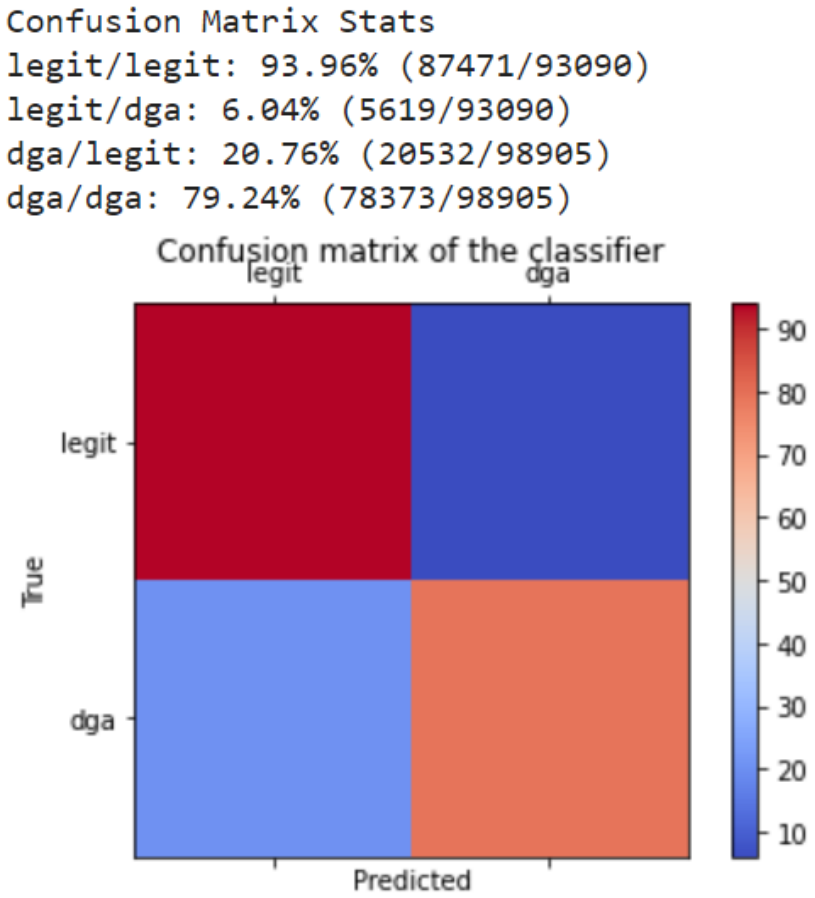



**Figure 5.13:** Confusion matrix of the second validation of the Long Short-Term Memory Model

than the XG-Boost. It must be said that the LSTM Model sees its performances drop by a greater percentage with respect to the two classifiers, as it starts from better results in the first analysis.

### 5.3.2 Bidirectional LSTM Model

The results of the first simulation for the Bidirectional Long Short-Term Memory Model are shown in Figure 5.14 for the classification report and in Figure 5.15 for the confusion matrix.

The overall Accuracy of the Bidirectional LSTM Model for the first validation is slightly less than 99%. As for the LSTM Model, the accuracy tells that also this approach has very high performances, but, as said before, this metric alone is not very useful and exhaustive. So, a deeper analysis can help in highlighting the differences of the models that the accuracy alone is not able to show.

As shown in the classification report in Figure 5.14, the values of the Precision for the two classes are comparable to the overall Accuracy, meaning that the number of mis-classified domain names is quite low for both the two classes. The Precision of the DGA class is higher than the one of the legit class (slightly more than 99%), showing that the number of mis-classified valid domain names is lower than the mis-classified DGA-based ones.

Also the Recall has very high values for both classes, around 98% for the DGA class and 99% for the legit one. This means that also the number of False Negatives is very low in both cases. Finally, the F1-Score has values similar to the Precision and the Recall, around 98.5% for the two classes.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.9807 | 0.9909 | 0.9858 | 93090 |
| 1 | 0.9914 | 0.9816 | 0.9865 | 98912 |
| accuracy | | | 0.9861 | 192002 |
| macro avg | 0.9860 | 0.9863 | 0.9861 | 192002 |
| weighted avg | 0.9862 | 0.9861 | 0.9861 | 192002 |

**Figure 5.14:** Classification report of the first validation of the Bidirectional LSTM Model

The analysis of the metrics tells that this Model has quite perfect performances, but with the study of the percentages of TP, FP, TN and FN, shown in the confusion matrix in Figure 5.15, the behavior of the Bidirectional Long Short-Term Memory Model is explained in details, highlighting the results of the single cases.

The first thing that is shown is that the Bidirectional LSTM Model performs

very well in correctly classifying the domain names. The very high percentage of True Negatives (slightly more than 99%) shows that the Model is close to perfection in understanding which are real domains, but it has very high performances also in the detection of the DGA-based ones, with the percentage of True Positives that is more than 98%.

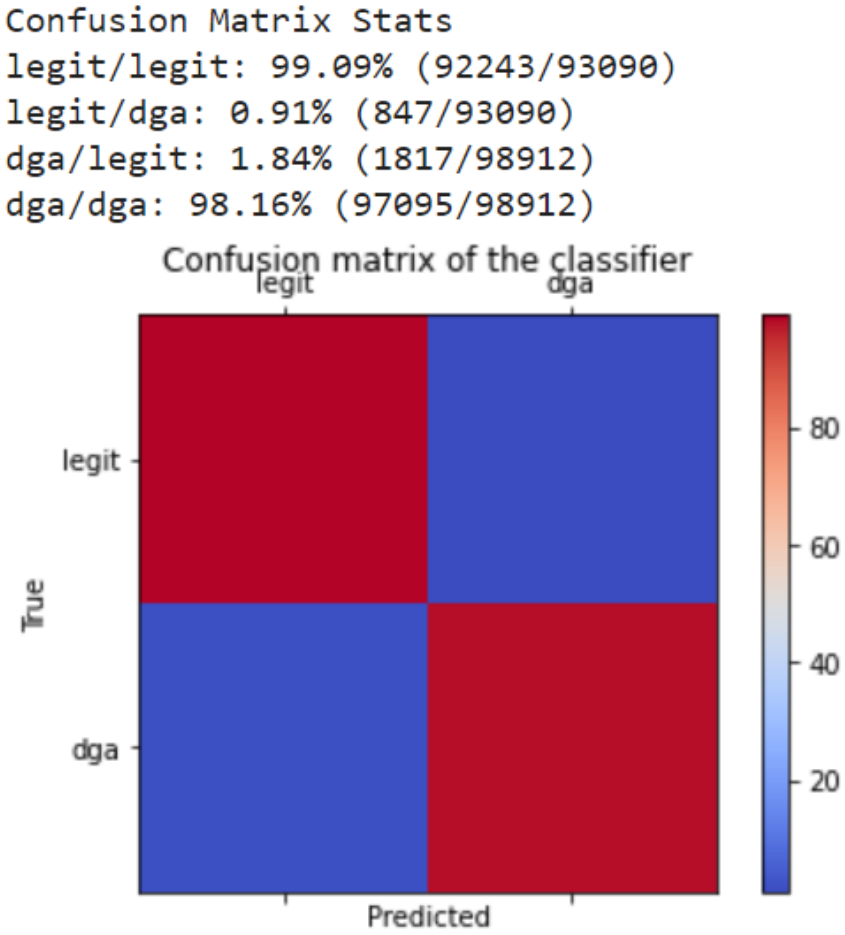



**Figure 5.15:** Confusion matrix of the first validation of the Bidirectional LSTM Model

Consequently, the percentages of False Positives and False Negatives are opposite, with slightly less than 1% of FP and slightly more than 1% of FN. This shows that the number of legit domain names labelled as anomalies that is fewer with respect

to the number of DGA-based ones labelled as valid. So the Model is very good in raising an alarm only in case of real anomalous domain names, with very few false alarms.

After having analyzed the metrics and the percentages of TP, FP, TN, FN, the considerations that can be made are that the Bidirectional LSTM Model has quite perfect performances in the DGA Detection problem, both in detecting almost all anomalous domains and in raising few false alarms, but also that it has better performances with respect to the traditional Machine Learning classifiers and slightly better with respect to the LSTM Model.

The results of the second simulation of the Bidirectional Long Short-Term Memory Model, with a set of different DGA families that the algorithm as never seen used as testing set, are shown in Figure 5.16 for the classification report and in Figure 5.17 for the confusion matrix.

As for the previous methods, the first difference to highlight in the two validations is that the overall Accuracy drops from the 98% of the first simulation to around 87% of the second one, that is a value comparable to the one obtained from the previous three classifications.

The overall Accuracy is always the starting point for the comparison of the performances, but, going deeper in the analysis, the behavior of the model can be explained in a better and exhaustive way, highlighting the reasons of the differences.

As shown in the classification report in Figure 5.16, the values of the Precision of the two classes are very different, 82% and 93% for the legit domain names and for the DGA-based ones respectively. This shows that the number of DGA-based domain names that are labelled as legit is very high, while there is a few number of valid domains labelled as anomalous, with results slightly less good than the ones of the first simulation.

Also the Recall has different values for the two classes, the opposite of the Precision, with 93% for the legit one and 80% for the DGA one, value that explains

that the number of False Negatives is quite high. Finally, because of the number of False Negatives of the two classes, the F1-Score values drops too, with values that are around to 87%.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.8211 | 0.9383 | 0.8758 | 93090 |
| 1 | 0.9329 | 0.8076 | 0.8658 | 98905 |
| accuracy | | | 0.8710 | 191995 |
| macro avg | 0.8770 | 0.8730 | 0.8708 | 191995 |
| weighted avg | 0.8787 | 0.8710 | 0.8706 | 191995 |

**Figure 5.16:** Classification report of the second validation of the Bidirectional LSTM Model

The analysis of the metrics has told interesting things, but the study of the percentages of TP, FP, TN and FN, shown in the confusion matrix in Figure 5.17, describe the behavior of the Bidirectional Long Short-Term Memory Model in a more detailed way.

The Bidirectional LSTM Model has very good performances in labelling correctly the legit domain names, but with results slightly lower to the ones of the first simulation. It is able to recognize with quite good results which are real domain names, as described by the percentage of True Negatives (slightly less than 94%) and a few number of False Positives (slightly more than 6%).

Even if the performances are decreased with respect to the first validation, there is only a few number of false alarms raised, which is always good in order to not signal everything as anomalous.

Also in this case, the number of False Negatives, in other words the DGA-based domain names that are labelled as valid, is the problem of the Bidirectional LSTM Model in this situation. The percentage of False Positives has increased up to around 19% and, consequentially, the number of True Positives has decreased, as shown by the percentage that has fell to slightly less than 81%.

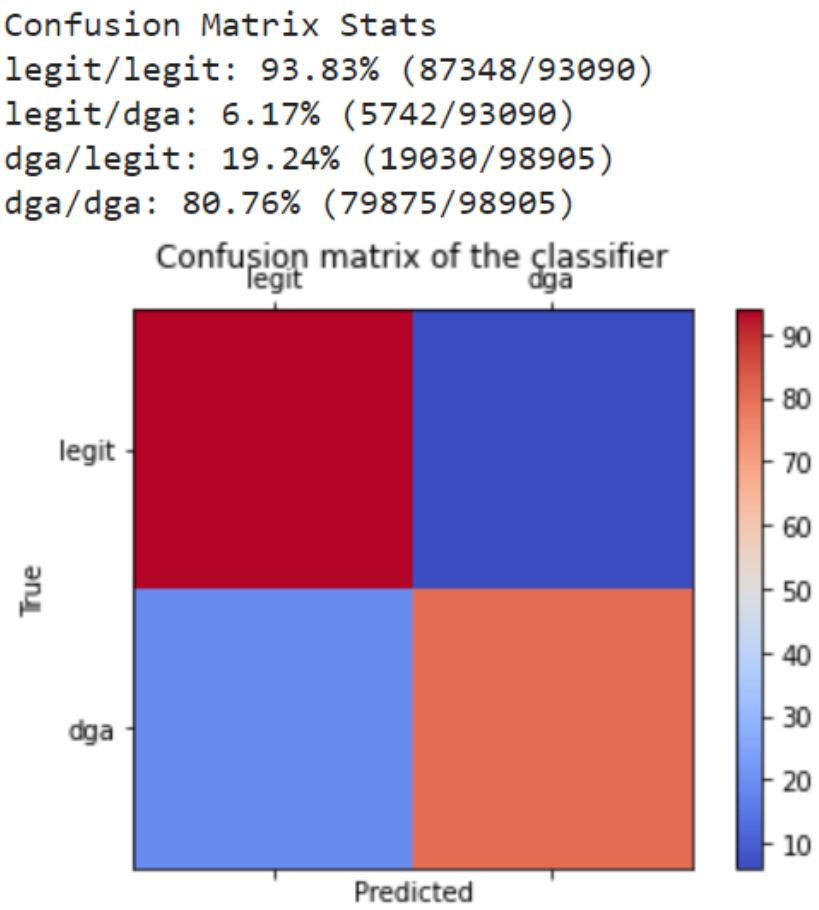



**Figure 5.17:** Confusion matrix of the second validation of the Bidirectional LSTM Model

These results tells that the Bidirectional LSTM Model has quite perfect performances when it has to label something that it has already seen in the training set. But, if it encounters something that it has never seen before, so if it is in the presence of new Domain Generation Algorithm families with different characteristics with respect to the ones that it knows, it makes it harder to detect them.

As for the LSTM Model, this problem tells that also the Bidirectional Long Short-Term Memory Model is over-fitting. It learns very well from the training set and it is quite perfect in classifying something similar to the things present in the training set, but it is focused only in the things that it knows, having difficulties in

label something it has never seen.

Making a first comparison of the Bidirectional LSTM Model with the two traditional Machine Learning Classifiers and the LSTM Model, it has better performances in the first validation case.

In the second one, it has performances similar to the ones of the Random Forest Classifier and LSTM, slightly less good that these two methods, but worst than the XG-Boost Classifier. Also the Bidirectional LSTM Model sees its performances drop by a greater percentage with respect to the two classifiers, but comparable to the LSTM Model behavior.

## 5.4 Results Summary

After having described the results obtained with the different models used, summarizing them is useful in order to have all the metrics and their values together and to make comparisons. Table 5.1 summarize the results obtained by the four models in the first validation, while in Table 5.2 there are the results obtained in the second one.

| Model | Accuracy | Precision avg | Recall avg | F1-Score avg | TP | FP | TN | FN |
|---|---|---|---|---|---|---|---|---|
| Random Forest | 93% | 93% | 93% | 93% | 92.67% | 6.19% | 93.81% | 7.33% |
| XG-Boost | 96% | 96% | 96% | 96% | 96.09% | 3.96% | 96.04% | 3.91% |
| LSTM | 98.65% | 98.64% | 98.66% | 98.65% | 98.36% | 1.04% | 98.96% | 1.64% |
| BiLSTM | 98.61% | 98.60% | 98.63% | 98.61% | 98.16% | 0.91% | 99.09% | 1.84% |

**Table 5.1:** Summary of the results of the first validation

| Model | Accuracy | Precision avg | Recall avg | F1-Score avg | TP | FP | TN | FN |
|---|---|---|---|---|---|---|---|---|
| Random Forest | 88% | 89% | 88% | 88% | 82.29% | 5.64% | 94.36% | 17.71% |
| XG-Boost | 92% | 93% | 93% | 92% | 88.61% | 3.60% | 96.40% | 11.39% |
| LSTM | 86.38% | 87.15% | 86.60% | 86.35% | 79.24% | 6.04% | 93.96% | 20.76% |
| BiLSTM | 87.10% | 87.70% | 87.30% | 87.08% | 80.76% | 6.17% | 93.83% | 19.24% |

**Table 5.2:** Summary of the results of the second validation

The first validation, as said before, was based on a random split of the dataset in training set and testing set. Analyzing the results of this first validation, it turns out that the two Deep Learning models have the best performances in terms of overall Accuracy (more than 98%), but also in terms of True Positives (more than

98%) and False Positives (around 1%), with the Bidirectional LSTM, as discussed, slightly better than the LSTM.

It must be said that the XG-Boost Classifier has very good results, with an overall Accuracy and True Positive Rate of 96% and False Positive Rate of less than 4%. This slightly lower results are balanced by a significantly faster training time. The Random Forest Classifier has the worst results, with an overall Accuracy around 93%.

The second validation, which tries to simulate the situation in which the models encounter something that they have never seen before (new kind of DGA families), gives interesting results to analyze. As expected, the four models lose some performance, because they can not learn something they have not seen yet, but the goal is to be robust and try to comprehend new anomalies.

It comes out that the best results are obtained with the XG-Boost Classifier, that has an overall Accuracy around 92%, the percentage of True Positives falls to slightly less than 89% and less than 4% of False Positives. There is a loss of performances, but the results remain quite good, demonstrating that the set of features extracted is very good in describing a domain name.

The two Deep Learning models have worse performances: the LSTM Accuracy fells to 86%, the number of True Positives to 79% and 6% of False Positives, while the Bidirectional LSTM Accuracy drops to 87%, slightly less than 81% of True Positives and 6% of False Positives. These results demonstrate that the two models are quite perfect in learning from the training set, but they are over-fitting, so they are not very strong in classifying something they have not seen before.

Even the Random Forest Classifier has better performances in the second simulation with respect to the LSTM and BiLSTM models. In fact, the overall Accuracy is around 88%, the True Positives percentage is 82% and the False Positives one is slightly less than 6%.

## 5.5 Performance

In order to analyze the performances of the methods in terms of execution time, the process has to be divided in pre-processing, model fitting, in other words the training time, and model evaluation.

The pre-processing phase of the two Deep Learing Models, that is the tokenization of the domain names takes few time, in the order of few minutes (3-4). The pre-processing phase of the traditional Machine Learning classifier is, obviously, the feature extraction process.

This process, made on the entire dataset, has taken a few minutes (10-15) for the extraction of all the features, except for the word breaker. The extraction of this feature has taken days, because it needed to split the domain name and compare it to sequences of English words.

This is no more a problem, because once the feature has been extracted for all the domain names in the dataset, the extraction of all the features on new domain names would take a few minutes.

The real difference between the proposed methods in terms of execution time is the model fitting phase. For the two traditional Machine Learning Classifiers, the Random Forest and the XG-Boost, the model fitting takes about 10-15 minutes.

This execution time obviously increases when there is a Neural Network model, which takes more time for fitting with respect to the traditional Machine Learning algorithms. The Long Short-Term Memory Model takes about 4 hours to complete the fitting phase, while the longest time for fitting is the one of the Bidirectional LSTM Model, that takes about 8 hours. So the Bidirectional LSTM takes more time for the training phase, as explained in Section 4.2.3.

The model evaluation phase, in other words the testing phase, takes more or less the same time for all the four methods used in this thesis work. They returned the results in times in the order of a few minutes (5-6).

# Chapter 6

# Conclusions and future developments

In this thesis work, different anomaly detection approaches were analyzed and studied in order to address the Domain Generation Algorithm detection problem. The anomaly detection methods used, that are based on classification problem solving, were divided in two macro approaches: the first one made of traditional Machine Learning algorithms and the second one made of Deep Learning models.

The first approach used in order to address the DGA detection problem, was based on a feature engineering process. With this process, many features was extracted from the domain names, with the goal to find the best set of features that represents the characteristics of the domains. This process was done with many attempts, trying different types of features, testing the classifiers with them and seeing which are useful and which are not.

After this phase of trials, the best set of features possible has been extracted, computed for every domain name in the dataset. Then, the classifier is trained with the training set and evaluated with a hold out set, the testing set. The two classifiers used for this approach were: Random Forest Classifier and XG-Boost Classifier.

The results of the validation of the two classifiers are obtained with two different situations, one with a random split of the dataset and the other with a set of different families as testing set, trying to simulate the situation in which the

algorithm encounters something that it has never seen before (a new kind of DGA family). Analyzing and comparing these results, described in Sections 5.2.1 and 5.2.2, it is clear that the XG-Boost Classifier has the best performances.

The second approach used in order to address the thesis experiment, was based on the use of the Neural Networks, that try to simulate the human brain behavior. The field for which the Neural Networks are used to solve the DGA detection problem is the text classification.

The idea is similar to the classification based on feature engineering of the first approach, studying the characteristics of the domain names, trying to find similar patterns that help to distinguish between real domain names and DGA-based ones.

This time the classification is not based on some pre-computed features, but on the domain name string itself, as can be seen from the name Text Classification. The Neural Networks learn patterns from the domain name and try to correctly distinguish them. The two Deep Learning model used are based on two different Neural Networks: the Long Short-Term Memory (LSTM) and the Bidirectional LSTM.

Also in this case, the models were evaluated on two situations: one with a random split and the other that simulate the case in which the model encounters new kind of DGA families. Comparing the results, described in Sections 5.3.1 and 5.3.2, the conclusion is that the two models have similar performances, with the Bidirectional LSTM slightly better than the LSTM, but with the drawback of the slower training time, as described in Section 5.5.

In order to find the best solution for the DGA Detection problem, the results and performances of all the four methods used are compared. It is necessary to analyze the models for the two simulation used.

After having analyzed the results of the four models in both validation cases, explained and described in Section 5.4, it turns out that the best solution, in terms of performances and time, is the XG-Boost Classifier. It has very high performances

in the first validation and, above all, it is the most robust model, in fact the results remains very good also in the second validation.

As said before, the XG-Boost Classifier is based on a set of extracted features. These ones were the best possible found during the thesis work, so possible future developments include, clearly, the addition of new features that can help the classifier in distinguish real and DGA-based domain names, or in improving the ones already present. For example, the starts_with_word feature could be replaced with a contains_word feature, that tells if the domain name contains an English word, not only if it starts with one of them.

In the Natural Language Processing (NLP) field, nowadays, transformers are used a lot. A transformer is, in simple words, a kind of Deep Learning model which adopts the self-attention mechanism, weighting in a different way the significance of each part of the input data. They are designed in order to process sequential input data like natural language.

In this thesis project, they were not analyzed because other NLP strategies have been used, but of course, transformers could be a possible future implementation for the text classification approach, so the one based on Deep Learning models.

Another possible future development is trying to combine the results of the proposed models, for example with a "voting" mechanism, in which, if the majority of the models label a domain name as anomalous, then the global algorithm raises an alarm, otherwise not.

Staying focused on the traditional Machine Learning models, based on feature extraction, the best thing to do is to update the dataset of the DGA-based domain names with the most recent ones, making sure that the classifier will learn new domain names characteristics.

# Bibliography


[1] UnderDefense. «Detecting DGA Domains: Machine Learning Approach». In: (Feb. 2021) (cit. on p. 5).

[2] *DGA Detection with Spark MLlib. URL:* URL: https://github.com/hmaccelerate/DGA_Detection (cit. on p. 6).

[3] *Spark MLib library. URL:* URL: https://spark.apache.org/mllib/ (cit. on p. 6).

[4] Fangli Ren, Zhengwei Jiang, Xuren Wang, and Jian Liu. «A DGA domain names detection modeling method based on integrating an attention mechanism and deep neural network». In: 3 (Feb. 2020). URL: https://doi.org/10.1186/s42400-020-00046-6 (cit. on p. 6).

[5] Kate Highnam, Domenic Puzio, Song Luo, and Nicholas R. Jennings. «Real-Time Detection of Dictionary DGA Network Traffic Using Deep Learning». In: 2 (Feb. 2021). URL: https://doi.org/10.1007/s42979-021-00507-w (cit. on p. 6).

[6] Savio Sciancalepore, Juhong Namgung, Siwoon Son, and Yang-Sae Moon. «Efficient Deep Learning Models for DGA Domain Detection». In: (Jan. 2021). URL: https://doi.org/10.1155/2021/8887881 (cit. on p. 6).

[7] *Alexa top 1M. URL:* URL: https://s3.amazonaws.com/alexa-static/top-1m.csv.zip (cit. on p. 8).

[8] *Netlab-360 DGA domains list. URL:* URL: https://data.netlab.360.com/dga/ (cit. on p. 8).

[9] *Python. URL:* URL: https://www.python.org/ (cit. on pp. 24, 37).

[10] *Pandas library. URL:* URL: https://pandas.pydata.org/ (cit. on pp. 24, 37).

[11] *NumPy library. URL:* URL: https://numpy.org/ (cit. on pp. 24, 37).

[12] *Matplotlib library. URL:* URL: https://matplotlib.org/ (cit. on pp. 24, 37).

[13] *Scikit-learn library. URL:* URL: https://scikit-learn.org/stable/ (cit. on pp. 24, 37).

[14] *Public Suffix List library. URL:* URL: https://pypi.org/project/publicsuffixlist/ (cit. on pp. 24, 37).

[15] *XG-Boost library. URL:* URL: https://xgboost.readthedocs.io/en/stable/ (cit. on p. 24).

[16] *Keras library. URL:* URL: https://keras.io/ (cit. on p. 37).