Detecting DNS Threats: A Deep Learning Model to Rule Them All

Franco Palau[1], Carlos Catania[1], Jorge Guerra[1], Sebastian Garcia[2], and Maria Rigaki[2]

[1] LABSIN, Facultad de Ingenieria. UNCuyo. Mendoza. Argentina
palaufranco12@gmail.com|harpo@ingenieria.uncuyo.edu.ar|jr@uncuyo.edu.ar
[2] CTU - Czech Technical University. Prague. Czech Republic
sebastian.garcia@agents.fel.cvut.cz

Abstract. Domain Name Service is a central part of Internet regular operation. Such importance has made it a common target of different malicious behaviors such as the application of Domain Generation Algorithms (DGA) for command and control a group of infected computers or Tunneling techniques for bypassing system administrator restrictions. A common detection approach is based on training different models detecting DGA and Tunneling capable of performing a lexicographic discrimination of the domain names. However, since both DGA and Tunneling showed domain names with observable lexicographical differences with normal domains, it is reasonable to apply the same detection approach to both threats. In the present work, we propose a multi-class convolutional network (MC-CNN) capable of detecting both DNS threats. The resulting MC-CNN is able to detect correctly 99% of normal domains, 97% of DGA and 92% of Tunneling, with a False Positive Rate of 2.8%, 0.7% and 0.0015% respectively and the advantage of having 44% fewer trainable parameters than similar models applied to DNS threats detection.

Keywords: Network Security · Botnet · Deep Neural Networks

1 Introduction

The Domain Name System (DNS) is a hierarchical and decentralized naming system for computers services. The main use of the DNS is to translate from Internet Protocol (IP) addresses to human-friendly names.

Despite DNS is not intended for a command channel or general purpose tunneling, during the past years several approaches have been developed to misuse it. In particular, approaches based on **DGA** (Domain Generation Algorithm) and **Tunneling**. DGA is an algorithm to generate domain names in a deterministic but seemly random way. Malware use DGAs to generate the next domain to access the Command & Control (C&C) communication server of a group of infected computers. On the other hand, since DNS is central to Internet, their ports are often left open by system administrators which facilitates the misuse of the system for establishing unauthorized Tunneling connections through the DNS.

A common approach for detecting DNS threats are the so-called lexicographical approaches, under such approaches domains are classified by studying the statistical properties of the characters conforming the domain name. Such as the frequency distribution of properties, domain character length, Shannon entropy, the vowel/consonant ratio and dictionary-based similarity among others [2]. Alternatively, Natural Language Processing (**NLP**) emerged as one of the most useful techniques for detecting DGA, especially in the analysis of the n-gram frequency distribution of domain names. An n-gram is defined as a contiguous sequence of n items from a given sequence of text. It is possible to use greater values for n than 1. In the simpler form, when $n = 1$, the single character frequency distribution is generated. The assumption is that DGA and Tunneling domains will have a different n-gram distribution than normal domain names.

Recently, to avoid the need for designing the right set of features for training machine learning classifiers, some authors explored the application of Deep Learning (DL) techniques. In particular the application of Long-Short-Term-Memory (**LSTM**) networks [7] and Convolutional Neural Networks (**CNN**) [3] for detecting DGA.

Since both DGA and Tunneling showed domain names with observable lexicographical differences with normal domains, it is reasonable to apply the same detection approach. Therefore, instead of training separate models for detecting DGA and Tunneling, the hypothesis of this work is that it is possible to build a network architecture capable of detecting both DNS threats. In particular, the detection mechanism was based on the application of a well-known Deep Learning technique: A Multi-Class Convolutional Neural Network (MC-CNN). When applied to the text analysis problem, the internal design of neural network cell is capable to capture combinations of characters that are important to discriminating malicious domains from non-malicious domains. This flexible architecture generalizes manual feature extraction like n-grams, but instead, it learns dependencies of one or multiple characters.

The main contributions of the present article are:

- A multi-class deep convolutional neural network capable of detecting DGA and Tunneling in Domain names.
- A detailed evaluation of MC-CNN on a extended dataset that includes domain names from 51 different real malware DGA and 3 commonly used Tunneling tools.

The rest of this paper is organized as follows: Section 2 provides a background, Section 3 describes the network architecture, Section 4 illustrates the experimental setup while Section 5 details the results, and finally in Section 6, we present the conclusions.

2 Background

Before presenting the neural network and its corresponding analysis, we provide basic information to understand the purposed of DNS threats and how they work. First, we introduce the concept of Botnets and why an attacker needs to communicate with them. Afterwards, we explain the DNS and how it is exploited by the attacker through DGA or Tunneling to establish a communication with botnets. Finally, we review how deep learning techniques are applied to detect these threats.

A Botnet can be conceived as a group of compromised computers which can be controlled remotely by an attacker. Generally, an attacker creates a botnet using malware that infects a large number of machines. Computers that are part of the botnet, are called *bots* or *zombies*. In order to achieve its goals, malware inside each bot needs to communicate with a *Command and Control* (C&C) *server*. A C&C server is a computer controlled by the attacker and serves as the headquarters for compromised machines in a botnet. Therefore, establishing C&C communications is a vital step for attackers to move laterally inside a network. Once the communication is established, malware can send stolen data, status information, receive instructions or even update itself to a new version. To achieve this connection, attackers take advantage of the DNS. A DNS server runs a protocol in charge of mapping more readily validated domain names into the numerical IP addresses needed for locating and identifying computer services and devices. By providing a worldwide, distributed directory service, the DNS has been an essential component of the functionality of the Internet.

When a hard-coded domain name is included inside malware to connect with the C&C server, reverse engineering can be used to find the associated IP address of the C&C center and thus, shut it down. To avoid this situation, attackers make use of DGA to generate a large number of pseudo-random domain names. A small subset of these domains are registered by the attacker, so at resolving these few domains, the malware will obtain a valid IP address and will communicate with the C&C center. One of the main advantages of this approach (from the attacker view) is that, by periodically changing the source of randomness and inputting it into the DGA, completely new domains can be generated. Therefore, detection approaches that relied on static domain blacklists were rapidly rendered ineffective.

Another technique attackers use to exploit the Domain Name System is **DNS Tunneling**. With this method, data is encoded inside DNS queries and responses, allowing attackers to exchange information in an obfuscated way. Since DNS was originally made for name resolution and not for data transfer, its often not seen as a malicious communications and data exfiltration threat. A requirement to use DNS Tunneling is that the attacker must control a domain and a DNS server, which receives the DNS requests for the domain and act as an authoritative server for that domain in order to run the server-side tunneling and decoding programs. By doing so, the attacker can observe all incoming DNS queries and have control under the answers to the queries. Therefore, the attacker can receive and send data, i.e. a two way communication channel. The advantages of DNS Tunneling include that DNS is almost always available, no direct connection is established between the bot and attacker, and pure data exfiltration (upstream only) is difficult to detect.

Catching these DNS threats has become a central topic in network security, leading to a recent interest in detecting malicious domains using machine learning techniques [14]. Precisely, Deep Learning (DL) models, trained solely on the domain name string, that classify domain names as benign or malicious have proven to be well-suited for this task. In [13], an LSTM network was presented as a DGA classifier reaching a 90% detection rate. With a similar focus, in [3] a DGA classifier was built, but in this case based on a simple 1D-CNN network capable of detecting more than 97% of total DGA domains with a false positive rate close to 0.7%. These DL classifiers have been built to identify only one type of DNS threat. We broaden this concept by presenting a multi-class classifier capable of detecting DGA and Tunneling in domain names with an architecture that requires fewer trainable parameters than similar networks applied to DNS threats detection. The following sections detail the deep learning approach presented in this article to achieve this task.

3 Deep Neural Network Architecture

As we mentioned in Section 2, our approach presents a deep neural network capable of detecting DGA and Tunneling in domain names. A simplified view of the architecture of the proposed solution is shown in Fig. 1. The Neural Network Architecture (**NNA**) model consists of an Embedding layer, a Conv1D layer, and finally 4 Dense fully connected layers. The input of the neural network consists of a domain name, that is pre-processed using tokenization based on valid domain name characters so that each character is represented by a unique integer. The first two layers are responsible for learning the feature representation in order to feed a traditional Multilayer Perceptron Network (MLP). This MLP consists of 4 fully connected layers (Dense layers). The first three layers use rectified linear unit (ReLu) as activation function, while the last one employs a softmax activation function in order to output the probability that a considered domain belongs to each of the three different classes considered. Beside the layers previously described, the complete NNA includes some other layers for dealing with the dimensions output of the Conv1D layer. It also includes a Dropout to prevent overfitting. Listing 1 details the implementation of the complete model in Keras, a well-known deep learning library [4] for Python Language.

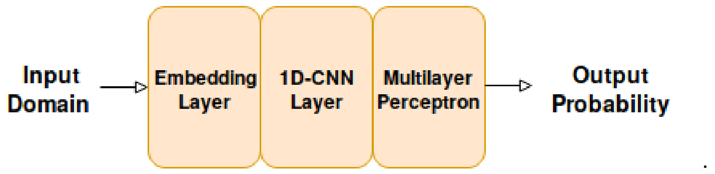

Fig. 1. Simplified View of Multi-Class Neural Network Architecture

```python
def multiclass_model(input_shape=(45,45), filters, size_kernel,
 ↪  dense1_size, dense2_size, dense3_size):
    model = Sequential()
    model.add(Embedding(input_dim=input_shape[0],
                        output_dim=100, input_length=input_shape[1]))

    model.add(Convolution1D(nb_filter=filters,
                            kernel_size=size_kernel,
                            strides=1,
                            activation='relu'))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(dense1_size,activation='relu'))
    model.add(Dense(dense2_size,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(dense3_size,activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='Adam',
                   ↪  metrics=[sparse_categorical_accuracy])
    return model
```

Listing 1: Code of Model Architecture implemented in Keras Library

4 Experimental Setup

In this section, we describe the dataset preparation along with the selected metrics and the hyper-parameters fine tuning methodology for improving model results.

4.1 Metrics

Several standard metrics are used for evaluating the network. These metrics are **Precision**, **Recall** or True Positive Rate (**TPR**), False Positive Rate (**FPR**) and **F1-Score**. Precision is computed as the ratio of items correctly identified as positive out of total items identified as positive while Recall is computed as the ratio of items correctly identified as positive out of total true positives. The F1-Score is the harmonic mean of Precision and Recall. Finally, False Positive Rate is computed as the ratio between items incorrectly identified as positive and the total number of actual negative predictions. These metrics are calculated for each class, i.e. Precision, TPR, FPR and F1-Score are calculated for normal domains, then DGA domains and finally for DNS Tunneling.

The results over classes are averaged using both micro and macro average. A micro-average calculates metrics globally by counting the total true positives, false negatives and false positives. This means that smaller classes will account for less in the average than larger classes. On the other hand, a macro-average will compute the metric independently for each class and then take the average. Hence, treating all classes equally. Since this paper focuses on the classification performance of the neural network for each class and not globally, macro average was chosen as the metric for experiments. However, both measures are provided for completeness.

Not only we will evaluate the performance of the network in terms of the metrics described before, but also we will take into consideration the number of trainable parameters in the model. By reducing these trainable parameters, the retrain time of the network is also reduced. This will allow the model to quickly adapt to the high variability present in real-time network traffic.

4.2 Dataset Description

The dataset used in this paper contains Normal, DGA and Tunneling domain names. Table 1 summarizes the total amount of examples for each domain type. The total number of normal domains are conformed by the Alexa top one million domains, 3,161 normal domains provided by the Bambenek Consulting feed and another 177,017 normal domains. In the case of DGA domains, they were obtained from the repositories of DGA domains of Andrey Abakumov [1] and John Bambenek [9], corresponding to 51 different malware families. For DNS Tunneling, 8000 tunnel domains were generated using the following set of well-known DNS tunneling tools under laboratory conditions: iodine [8], dnscat2 [5] and dnsExfiltrator [6]. The complete dataset can be found in [11].

Table 1. Total Amount of Examples for Each Domain Type

Domain Type	**Number of Examples**
Normal	1,180,178
DGA	1,915,335
Tunneling	8,000

As can be noticed in Table 1, the number of Normal or DGA domains far outweigh the number of Tunneling domains. This means that the dataset is highly imbalanced.

The complete dataset detailed in this section was split into two new datasets: the first one, containing 70% of total entries, is used for the training and tuning of the network. The second one, containing the remaining 30% of total entries, is used for the evaluation of the proposed model on unseen domains.

4.3 Hyper-Parameters Model Tuning

The proposed model possesses many hyper-parameters that need to be tuned. In order to complete the fine tuning task, a traditional Grid Search was conducted. In the case of the CNN layer, we focused on optimizing the number of filters and the kernel size, while for the MLP, the size of each dense layer was chosen. Finally, the fraction rate of Dropout was also adjusted. Table 2 illustrates each hyper-parameter with its corresponding value range and the optimal value encountered. The tested values for the dense layers size and the number of filters follow a 2^n distribution, where the range of n varies according to each hyper-parameter. Tested values for Kernel size and Dropout rate increased linearly. Finally, a K-fold cross validation was performed with K=5 folds for a better estimation of each parameter. The training process was carried out during 10 epochs, using the Adaptive Moment Estimation optimizer [10] and backpropagation algorithm [12].

Table 2. Values for Hyper-parameter Tuning

Hyper-parameter	**Value Range**	**Optimal Value**
Number of filters	64 - 1024	512
Kernel size	4 - 8	4
First dense layer size	256 - 1024	256
Second dense layer size	256 - 1024	1024
Third dense layer size	256 - 1024	256
Dropout rate	0.2 - 0.8	0.5

Since many combinations of hyper-parameters are evaluated, the tuning becomes costly in terms of time. To reduce this time, instead of training each model with the complete dataset, only a fraction of the dataset was used. To ensure that this subset is representative of the whole data, we analyzed the Frequency Character Distribution (**FCD**) of each domain type.

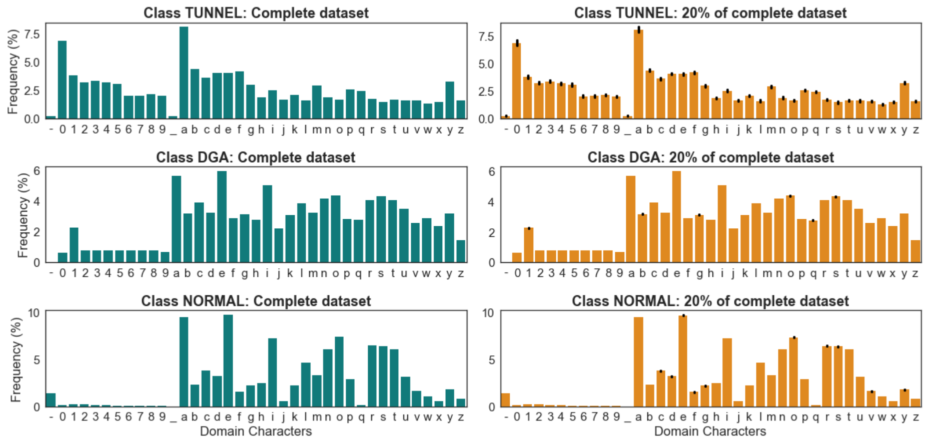

Fig. 2. Frequency Character Distribution of total dataset versus 20% of total dataset discriminated by domain type

As can be seen from Fig. 2, when a fraction of 20% is selected, the FCD of each domain type in the subset is almost exactly the same as if all data were considered. Due to domains that conform the subset are chosen randomly from the complete dataset, the experiment was repeated 30 times and the average and standard deviation are plotted in FCD chart. The optimal combination of parameters after tuning with the illustrated fraction of total domains can be found in Table 2.

5 Results

In this section, we present the evaluation of the proposed model on unseen domains. Then, considering that the purpose of this model is to detect normal, DGA and Tunneling domains, we compare the performance of the MC-CNN model against two state-of-the-art binary classifiers, one made for DGA detection and the other one for Tunneling detection.

5.1 Performance of MC-CNN

The hyper-parameters selected in the previous section were employed for training the network. In this case, the complete training dataset was used. Then, the model was tested using the explained testing dataset. The experiment results are shown in Figure 3 with the resulting Confusion Matrix. Based on this, metrics illustrated in Section 4.1 are calculated and described in Table 3. The support (size of test set) is given in the last column.

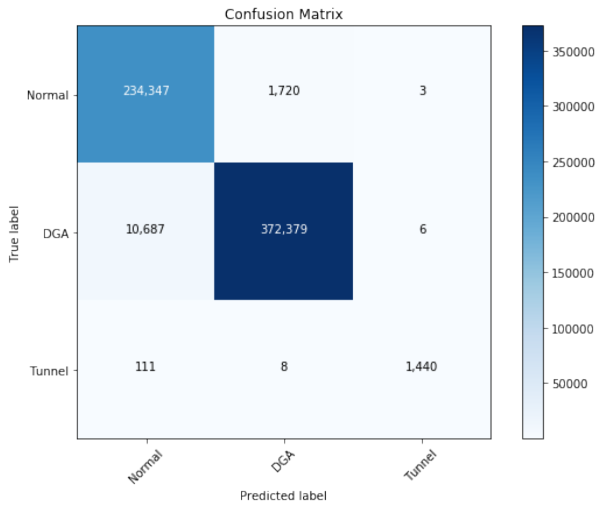

Fig. 3. Confusion matrix

Overall, the MC-CNN has an outstanding performance reaching a macro average F1-score of 97%. Although the dataset is highly imbalanced, the resulting model is able to detect correctly 99% of total normal domains, 97% of total DGA and 92% of total Tunneling, with FPR of 2.8%, 0.7% and 0.0015% respectively.

Table 3. Metrics resultant on testing with 30% of the dataset

Domain Type	**Precision**	**Recall (TPR)**	**FPR**	**F1-Score**	**Support**
Normal	0.96	0.99	0.028	0.97	236072
DGA	1.00	0.97	0.00727	0.98	383072
Tunneling	0.99	0.92	1.45e-05	0.96	1559
macro average	0.98	0.96	0.01	0.97	620703
micro average	0.98	0.98	0.01	0.98	620703

5.2 Binary Classifiers for Detecting DNS Threats

We compare the proposed multi-class model against two binary classifiers (BC): one, for identifying DGA and normal domains (**BC-DGA**), and the other one for Tunneling and normal domains (**BC-Tunnel**). As described in [3], the BC-DGA proposed for detecting DGA and normal domains was trained with the same DGA and normal domains described in section 4.2. The network architecture for this classifier is shown in Listing 2. This binary classifier achieved a TPR of 97% and a FPR of 0.7% in terms of DGA detection. For the BC-Tunnel model, we used the same architecture and combination of hyper-parameter shown in Listing 2, but in this case, instead of training with DGA domains, Tunneling domains were used. The resulting tunnel-normal classifier had a TPR of 94% and a FPR of 0.0017% in terms of Tunneling detection.

Table 4 details the metrics illustrated in Section 4.1 for the BC-DGA model, the BC-Tunnel model and the proposed MC-CNN model. It can be clearly seen that, with the proposed multi-class model in this paper, we were able to replicate almost the exact same performance as with both binary classifiers in terms of DGA and Tunneling detection.

Table 4. Comparison of Binary Models and Multi-class Model

Domain Type	**Recall (TPR)**			**FPR**			**F1-Score**		
	BC-DGA	BC-Tunnel	MC-CNN	BC-DGA	BC-Tunnel	MC-CNN	BC-DGA	BC-Tunnel	MC-CNN
Normal	0.99	0.99	0.99	0.025	0.404	0.028	0.97	0.99	0.97
DGA	0.97	-	0.97	0.007	-	0.007	0.98	-	0.98
Tunneling	-	0.94	0.92	-	1.69e-5	1.45e-5	-	0.97	0.96

```python
def binary_model(input_shape):
    model = Sequential()
    model.add(Embedding(input_dim=input_shape[0], output_dim=100,
     ↪  input_length=input_shape[1]))

    model.add(Convolution1D(nb_filter=256,
                            kernel_size=4,
                            strides=1,
                            activation='relu'))
    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer='adam', metrics=['accuracy'])
    return model
```

Listing 2: Neural Network Architecture for Binary Classification

Although the architecture implemented for both binary classifiers has fewer layers than the multi-class classifier, the model built for the binary classifiers requires to train a total of 5,613,205 parameters for each model, while the multi-class model has a total of 6,241,431 trainable parameters. This means that solving the problem of identifying Normal, DGA and Tunneling domains with two binary classifiers would require to train more than 11 million parameters. On the other hand, with the proposed multi-class model, we were able to solve this problem reducing the number of trainable parameters by 44% and maintaining almost the same performance.

6 Concluding Remarks

In the present work, we explore the viability of a MC-CNN for lexicographical DNS threats detection. We evaluated the proposed model on a dataset containing 2 types of DNS threats: DGA and Tunneling, as well as normal domains. The dataset was properly split into training and testing sets.

We compared the FCD of the complete dataset versus only a certain fraction of total domains, showing that we were able to reproduce almost the exact same FCD of the complete dataset with only a 20% of total domains. Therefore, we conducted a hyper-parameters grid search on this fraction set and thus, reducing tuning time. The best-resulting model was then evaluated on testing set. The resulting MC-CNN is able to detect correctly 99% of total normal domains, 97% of total DGAs and 92% of total Tunneling, with a FPR of 2.8%, 0.7% and 0.0015% respectively.

The performance of the MC-CNN was compared against two binary classifiers comprised of a simple 1D-CNN network architecture: one in charge of detecting DGA and the other for detecting Tunneling. The experiments showed that the

MC-CNN reproduced almost the same performance as the binary classifiers in terms of DGA and Tunneling detection, with the advantage of having to train 44% fewer parameters in order to detect all these threats.

Although the dataset is highly imbalanced for Tunneling domains, the MC-CNN was able to detect them with an F1-score similar to the other domain types. It is possible that this behaviour is due to the fact that the FCD of the three types of domains considered are very different from each other. Moreover, the Tunneling domains considered in this work were generated using only 3 different tools. Therefore, it could be necessary an in-depth analysis of how including more tunneling tools affects the performance of the network.

Acknowledgments

The authors would like to thank the financial support received by CVUT and UNCuyo during this work. In particular the founding provided by the Czech TACR project no. TH02010990 and the PICT 2015-1435 granted by ANPCyT. The authors would also like to specially thank Whalebone s.r.o., whose technical support and help have been fundamental to the complete research process. In addition, we want to gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

1. Andrey Abakumov: Repository containing dga domain names. `https://github.com/andrewaeva/DGA`
2. Antonakakis, M., Perdisci, R.: From throw-away traffic to bots: detecting the rise of dga-based malware. Proceedings of the 21st USENIX Security Symposium p. 16 (2012)
3. Catania, C., Garcia, S., Torres, P.: An analysis of convolutional neural networksfor detecting dga (2018). https://doi.org/10.13140/RG.2.2.11771.16165
4. Chollet, F., et al.: Keras. `https://github.com/fchollet/keras` (2015)
5. Dnscat2: Tool for creating an encrypted command-and-control channel over the dns protocol, `https://github.com/iagox86/dnscat2`
6. DnsExfiltrator: Data exfiltration over dns request covert channel, `https://github.com/Arno0x/DNSExfiltrator`
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), `http://www.deeplearningbook.org`
8. Iodine: Tool for tunneling ipv4 data through a dns server, `https://code.kryo.se/iodine/`
9. John Bambenek: Feed listing containing several dga families. `http://osint.bambenekconsulting.com/feeds/`
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014)
11. Palau, F., et al.: DNS Threats Dataset. `https://datahub.io/palaufranco12/asai-2019-multiclass/v/1` (2019)

12. Rumelhart, D.E., Hinton, G., Williams, R.: Neurocomputing: Foundations of research, chap. Learning Representations by Back-propagating Errors, pp. 696–699. MIT Press (1988)
13. Woodbridge, J., Anderson, H., Ahuja, A., Grant, D.: Predicting domain generation algorithms with long short-term memory networks (2016), http://arxiv.org/abs/1611.00791
14. Zago, M., Prez, M., Martinez Perez, G.: Scalable detection of botnets based on dga: Efficient feature discovery process in machine learning techniques. Soft Computing (2019). https://doi.org/10.1007/s00500-018-03703-8