Botnet Detection on TCP Traffic Using Supervised Machine Learning

Javier Velasco-Mata[12], Eduardo Fidalgo[12], Víctor González-Castro[12], Enrique Alegre[12], and Pablo Blanco-Medina[12]

[1] Department of Electrical, Systems and Automation Engineering, Universidad de León, Spain
[2] Researcher at INCIBE (Spanish National Cybersecurity Institute), León, Spain
{jvelm,eduardo.fidalgo,victor.gonzalez, enrique.alegre, pblanm}@unileon.es

Abstract. The increase of botnet presence on the Internet has made it necessary to detect their activity in order to prevent them to attack and spread over the Internet. The main methods to detect botnets are traffic classifiers and sinkhole servers, which are special servers designed as a trap for botnets. However, sinkholes also receive non-malicious automatic online traffic and therefore they also need to use traffic classifiers. For these reasons, we have created two new datasets to evaluate classifiers: the TCP-Int dataset, built from publicly available TCP Internet traces of normal traffic and of three botnets, Kelihos, Miuref and Sality; and the TCP-Sink dataset based on traffic from a private sinkhole server with traces of the Conficker botnet and of automatic normal traffic. We used the two datasets to test four well-known Machine Learning classifiers: Decision Tree, k-Nearest Neighbours, Support Vector Machine and Naïve Bayes. On the TCP-Int dataset, we used the F1 score to measure the capability to identify the type of traffic, i.e., if the trace is normal or from one of the three considered botnets, while on the TCP-Sink we used ROC curves and the corresponding AUC score since it only presents two classes: non-malicious or botnet traffic. The best performance was achieved by Decision Tree, with a 0.99 F1 score and a 0.99 AUC score on the TCP-Int and the TCP-Sink datasets respectively.

1 Introduction

The development of the Internet as a common resource has caused a constant grow of the number of cyberthreats. Therefore, the need of automatic solutions to supply the lack of human resources to detect these threats has fomented the use of Machine Learning (ML) algorithms in cybersecurity [1]. One of the most hazardous menaces are *botnets*.

A botnet is a network of malware-infected computer-like devices called *bots*, which are controlled by a remote user, known as *botmaster*. Botnets are used to perform different types of cyber-attacks such as DDoS (Distributed Denial of Service) to targeted servers or massive credential collection from online users. For

this reason, botnets have become one of the most remarkable security concerns in the last years [2, 3]. Depending on the structure of the network [2], botnets can be classified as:

- **Centralized:** In this structure all the infected devices communicate directly with the botmaster.
- **Distributed:** In this case, the botnet use peer-to-peer protocols for intra-communication and the botmaster only connects to a small subset of botnets to transmit the orders.
- **Hybrid:** This type of network is divided into subsets of bots. Each subset follows a centralized hierarchy where all the *client* bots communicate with a *server* bot. On a higher level, the server bots communicate with each other and with the botmaster using a distributed architecture.

Botnets need to receive orders from online connections, and the generated traffic can be exploited to detect them. Early solutions used a signature-based approach [4] and thus analysed plain network communications in search of known malicious patterns, but they can be bypassed using obfuscation or encrypted protocols. The supervised ML based detectors can overcome this drawback and provide a more efficient solution. A trained ML model is capable of classifying the network traffic according to general traffic characteristics instead of looking for specific signatures [5].

Beside of traffic classifiers, botnets also can be detected using *sinkhole servers*. A sinkhole is a server designed to deceive botnets by disguising as a real botnet controller server, and thus it collects the attempts of bots to communicate with the botmaster [6]. However, sinkholes also receive non-malicious connections from automatic programs like web crawlers or *spiders*, i.e., programs that systematically explore the Internet usually for web indexing [7]. Because of this, it is also necessary to apply network classifiers to the traffic received by sinkholes servers to separate non-malicious traces from the botnet ones.

This work explores the automatic classification of TCP network traffic, both in regular Internet communications and in sinkhole traffic. Besides, we have not found another work on botnet detection in sinkhole traffic focused on the TCP protocol.

We present two main contributions. First, we constructed two new datasets from TCP network traffic: one from public Internet traffic captures and another from private sinkhole data. And second, we tested on the two datasets four well-known ML classifiers, namely Decision Tree (DT), k-Nearest Neighbours (k-NN), Support Vector Machine (SVM) and Naïve Bayes (NB), using a scheme applicable to the three types of botnet architectures we previously described. The general scheme of this work is illustrated on Fig. 1.

The rest of the paper is organized as follows. Section 2 presents the literature review about botnet detection and classification. In Sect. 3 we present the datasets we made for this work, and we describe the features used to model the traffic samples, as well as the tested algorithms and the performance metrics we have used. The experimental settings are described on Sect. 4, and the results are discussed on Sect. 5. Finally, our conclusions and future work are on Sect. 6.

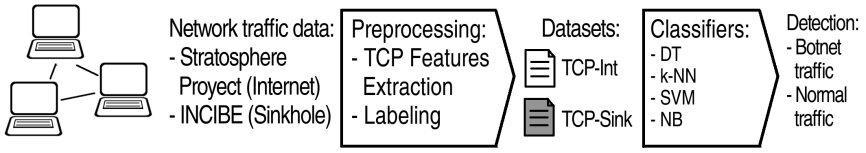

Fig. 1. Methodology followed in this work

2 Related Works

Over the last years, different works have proposed various ML algorithms to detect botnets with varying results. These algorithms have been compared in a variety of network traffic classification problems, such as detection of anomalies, network intrusions or botnet detection.

For example, the experiments of Kirubavathi and Anitha [5] used a mixture of the ISOT botnet dataset[3] and a dataset of private laboratory traces, obtaining an accuracy of 99.14% for NB while the accuracies for DT and SVM were 95.86% and 92.02% respectively.

However, Sangkatsanee et al. reported an accuracy on their work about intrusion detection [8] of 99.00% using DT, outperforming NB, Bayesian Networks (BN) and shallow Neural Networks (NN), which got an accuracy of 78.70%, 89.30% and 93.00% respectively. These results were obtained with a private dataset with 7200 data records. Based on these results, they implemented a DT in a real-time Intrusion Detection System (IDS) that showed a good performance, with a Total Detection Rate (TDR) of 99.33%.

Moreover, Kim et al. [9] concluded that SVM surpasses k-NN, NB, BN, DT and shallow NN in network anomaly detection with an accuracy comprised between 94.2% and 97.8% on their datasets of collected network traces from different entities in the USA, Japan and Korea.

Recently, the appearance of the Internet of Things (IoT) has offered another environment for botnet propagation. The research of Doshi et al. [10] on the Mirai botnet propagation over IoT devices demonstrated the effectiveness of five ML classifiers, namely k-NN, SVM, DT, Random Forest (RF) and NN, on self generated IoT traffic with Mirai traces. In particular, RF achieved the highest mean F1 score (0.99).

3 Methodology

3.1 Datasets and Features

The Internet-alike network traffic data used in this work is publicly available by courtesy of the of the Stratosphere Project, formerly Malware Capture Facility Project [11]. The traffic captures that we have selected for this work are listed on

[3] https://www.uvic.ca/engineering/ece/isot/datasets/

Table 1. Our dataset is composed of traces from these data where each sample corresponds to a TCP connection or *TCP flow* and is characterized with the features described in Table 2. The dataset, which we called *TCP-Int dataset*, contemplates four classes of data: the Normal class represents non-malicious traffic, while the Kelihos, Miuref and Sality classes represent traffic traces from the botnets with the same name. We built a balanced dataset where each class contains 44231 samples since this is the number of TCP connections of the least represented class, *Miuref*.

Table 1. List of the selected traffic captures including the type of their traffic (i.e., from Normal communications or from activity of the malware Kelihos, Miuref or Sality) and the number of TCP flows present. The source of the captures is the Stratosphere Project [11]

Traffic capture	Type	TCP flows
ctu-normal-21.pcap	Normal	5886
ctu-normal-22.pcap	Normal	45461
ctu-normal-23.pcap	Normal	4059
ctu-normal-24.pcap	Normal	1721
ctu-normal-25.pcap	Normal	1092
ctu-normal-26.pcap	Normal	1866
ctu-normal-27.pcap	Normal	9200
ctu-normal-28.pcap	Normal	4034
ctu-normal-30.pcap	Normal	13965
ctu-normal-31.pcap	Normal	14982
ctu-normal-32.pcap	Normal	22000
2015-12-09_capture-win4-1.pcap	Kelihos	95575
2015-12-09_capture-win4-2.pcap	Kelihos	94219
2015-06-07_capture-win12.pcap	Miuref	1285
2015-06-07_capture-win8.pcap	Miuref	4899
2015-06-19_capture-win12.pcap	Miuref	4807
2015-07-08_capture-win8.pcap	Miuref	33240
2014-04-07_capture-win13.pcap	Sality	57808

Besides, we have constructed a dataset using traffic data from a sinkhole server which we called *TCP-Sink dataset*. These data was supplied by INCIBE (Spanish National Cybersecurity Institute)[4], and have both non-malicious traces from automatic online programs and traces from the Conficker botnet. The TCP-Sink dataset contains 4027 samples of TCP flows from non-malicious connections (Normal class) and the same amount of samples from Conficker connections (Botnet class). The samples are described according to the TCP features shown in Table 2.

[4] https://incibe.es/

Table 2. List and descriptions of the TCP flow features used in this work

Feature(s)	Description
sPort, dPort [12, 13]	Source and destination ports in the TCP connection
mLen, vLen [12, 13]	Mean and variance of the payload lengths of all packets in the TCP connection
mTime, vTime [13]	Mean and variance of the interval times between sent packets in the TCP connection
mResp, vResp [5]	Mean and variance of the interval times between a packet reaches the machine that started the connection and it responds to that packet
nBytes [12]	Total number of bytes exchanged in the TCP connection
nSYN [13]	Total number of SYN flags exchanged during the TCP connection
nPackets [12]	Total number of packets exchanged in the TCP connection
mVel, vVel [13]	Mean and variance of the number of packets exchanged per second in the TCP connection

3.2 Classifiers

We have chosen four ML algorithms that achieve a high detection rate in other works [5, 8–10] on botnet detection to study their performance on our two datasets, the TCP-Int and the TCP-Sink ones. In spite of finding more algorithms in the network classification literature, for a fair comparison we have avoided comparing assembled methods with their non-assembled versions, for example Random Forest with Decision Tree.

Decision Tree (DT): This algorithm can be represented as a tree of logical decisions [14], where each decision splits the path in two or more [15]. A new query starts its route from the root, and depending on the result in each decision, it will follow a certain path. The class of the data is determined by the end of its path.

K-Nearest Neighbours (k-NN): Let each sample of the labelled training data has m features and thus, it can be represented as a labelled vector in a m-dimensional space. Given an input data sample, also represented as a m-dimensional vector, k-NN [16] searches for the k nearest vectors from the training data. The input sample is classified depending on the labels of these vectors, and it is regular to used an odd k value to avoid a tie between neighbours.

Support Vector Machine (SVM): Given that each sample from the training dataset has m features, it can be viewed as a point in a m-dimensional space. The SVM model [17] looks for the hyperplane or set of hyperplanes that best separates points from the same class from points of different classes. If necessary, SVM uses a function called *kernel* or *kernel function* that projects the points into a higher dimensional space where they can be separated more easily by a

hyperplane. Depending on the problem characteristics, SVM can use either a generic kernel such as the linear kernel [18] or a custom kernel [19–21].

Naïve Bayes (NB): Given a value x for a certain feature f, the probability that this value corresponds to a certain class C_j is $P(C_j|f = x)$. This probability is calculated by fitting the training data into a probability distribution defined by the event model of NB [22, 23]. To calculate the probability that an input data corresponds to a certain class, NB supposes that each feature contributes independently [24].

3.3 Evaluation

In this work we have used three metrics to measure the performance of the classification algorithms, which are described as follows:

F1 score: This score is defined as the harmonic mean of the precision and recall scores [25] as shown in (1). The precision is defined in (2) from the number of True Positives (TP) and the number of False Positives (FP), while the recall is defined in (3) from the number of TP and the number of False Negatives (FN). The F1 score can be applied with multiclass datasets to measure the performance of a classifier over each class, where the ideal score is 1 and the worst is 0.

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

Receiver Operating Characteristic (ROC) curves: ROC curves [26] usually plot the True Positive Rate (TPR) on the Y axis against the False Positive Rate (FPR) on the X axis using the results of cross-validation. This method is used to evaluate the performance of a classifier on datasets with two types of data such as the TCP-Sink dataset. The ideal measure maximizes the TPR while minimizing the FPR.

Area Under the Curve (AUC): This measurement corresponds with the total area under the ROC curve [26]. The ideal measure is an area of 1, which means a perfect TPR and FPR scores, while a score under 0.5 means that the performance is worse than a random classifier.

4 Experiments

All the experiments were performed on a machine with 128 GB of RAM and two Intel(R) Xeon(R) E5-2630v3 CPUs running an Ubuntu 14.04.5 LTS.

The software was developed using Python 3. To extract traffic characteristics from traffic captures in .pcap format, the script used the module PyShark[5] which is a python wrapper for TShark[6], and the module PyTables[7] to manage intermediary HDF5 files. The Machine Learning Python module used was sklearn[8], and the most relevant optimization settings for the four used algorithms in the sklearn libraries are described as follows:

1. For DT, the main setting is the maximum depth which refers to the maximum number of splits in a path from the root to a leaf. While reducing this depth allows decreasing the computational cost, the downside is a decrease on the detection capability. In this work we use the default option of unlimited depth, i.e., it is not established a restriction on the maximum number of branches in the tree.
2. In k-NN the main hyper-parameters are the number k of neighbours considered in a search and the distance metric used to determine the similarity with the neighbours. Before comparing k-NN with the rest of the classifiers, we experiment with different values of k to determine the optimal one for each of the two datasets used in this work. Besides, to determine the similarity with the neighbours, we use the Euclidean distance in the 11-dimensional space of the features from Table 2.
3. The main settings for the SVM algorithm are the employed kernel and the error penalty parameter or cost C. In this work we have chosen an optimised implementation of the linear kernel that uses the liblinear library [27] due to its better escalation on a large number of samples, which is beneficial on network traffic classification problems. With respect to the cost variable, we have tried several values (C=$\{0.1, 0.5, 0.9, 1.0, 5.0, 10.0, 100.0\}$) without observing any significant difference, and thus we have selected the default value of C=1.0.
4. The only relevant setting for NB is the employed event model, whose selection is based on the type of the used data. In this work, we use features represented as continuous variables that can not be converted into discrete ones, such as all the features on Table 2 that represent a mean or a variance, and thus we use a Gaussian event model.

We tested the four algorithms using stratified 10-folds cross-validation on the TCP-Int and the TCP-Sink datasets introduced on Sect. 3.1.

[5] https://pypi.org/project/pyshark/
[6] https://www.wireshark.org/
[7] https://www.pytables.org/
[8] http://scikit-learn.org/stable/index.html

5 Results and Discussion

We have tested the performance of the four classifiers on Internet-alike traffic using the TCP-Int dataset. This dataset presents four classes: Normal, Kelihos, Miuref and Sality, and we used the F1 score to measure the class-identification ability of each algorithm.

First, we need to find the optimal k parameter of the k-NN classifier. Figure 2 shows that the highest mean F1 score of k-NN on the TCP-Int dataset is achieved with $k = 1$.

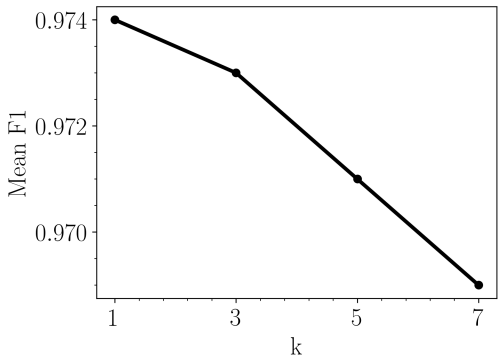

Fig. 2. Mean F1 score achieved by k-NN with different k values over the TCP-Int dataset

Afterwards, we tested the four classification algorithms (with $k = 1$ in k-NN) on the TCP-Int dataset and the results are displayed on Table 3. DT achieves the best detection rate with a mean F1 score of 0.99 with a stable detecting performance among the four classes, meaning that the current used features are easy to categorize. On the one hand, k-NN also presents a relatively high and stable performance, implying that the data is well separated into clusters on the feature space. On the other hand, SVM with a linear kernel shows a relatively poor detection rate over the Normal and Sality classes. This suggest that the Normal and Sality classes are not linearly separable in the feature space. Besides, NB only achieves a relatively good result on the Kelihos class, while it shows a poor performance on the other three classes with an approximate 50% F1 score. Since the NB classifier is based on the idea that the features are independent, we infer that the Normal, Miuref and Sality classes contain features that present correlations.

To test the performance of the four classifiers on sinkhole traffic we used the TCP-Sink dataset. Since this dataset only contains two classes, Normal and Botnet traffic, it is preferred to measure the performance using ROC curves and AUC scores because they are more informative in binary classification than the F1 score. First, we need to optimize the k parameter of k-NN. As depicted in

Table 3. F1 scores by class achieved by the four models tested over the TCP-Int dataset. †Appears as perfect score due to two decimal rounding

Class	Samples	DT	k-NN	SVM-linear	NB-gauss
Normal	44231	**0.99**	0.96	0.66	0.49
Kelihos	44231	**1.00**†	0.99	0.86	0.75
Miuref	44231	**1.00**†	0.98	0.83	0.51
Sality	44231	**0.99**	0.97	0.64	0.51
Mean		**0.99**	0.97	0.74	0.56

Fig. 3, we observe that the AUC score first increases with k and then stabilises after $k = 13$.

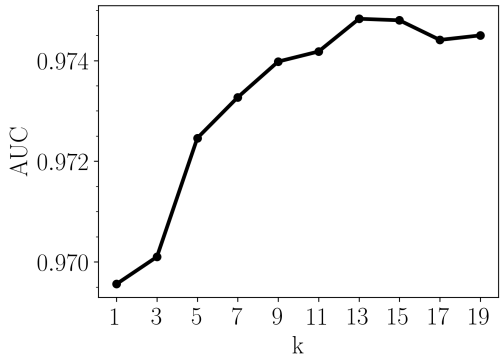

Fig. 3. AUC scores achieved by k-NN with different k values over the TCP-Sink dataset

After optimizing k-NN with $k = 13$, we compared the accuracy of the four classifiers on the TCP-Sink dataset to discern between non-malicious traces from Conficker traces. The achieved AUC scores are tabulated on Table 4 and the ROC curves are shown on Fig. 4. We notice that all the four classifiers achieve a high accuracy, where the best result is obtained by DT with an AUC of 0.99. Taking into account the poor performance of SVM and NB on the TCP-Int dataset and their good results on the TCP-Sink dataset, we deduce that the non-malicious traces from automatic connections of online services are easily identifiable from the Conficker botnet ones.

Table 4. AUC scores for binary classifications over the TCP-Sink dataset

Model	AUC
DT	**0.99**
k-NN	0.97
SVM-linear	0.89
NB-gauss	0.88

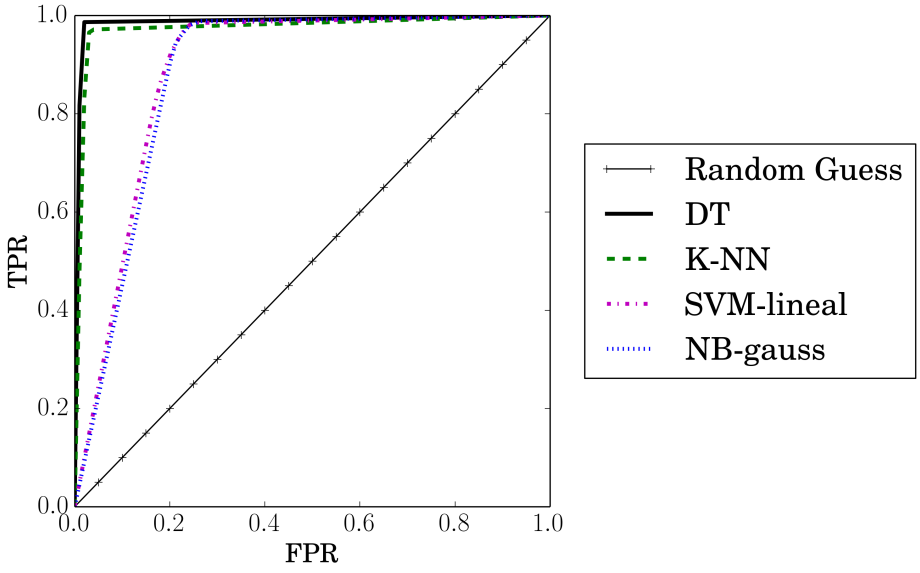

Fig. 4. ROC curves for Conficker botnet detection over the TCP-Sink dataset

6 Conclusions and Future Work

In this paper we explored the problem of botnet detection on TCP traffic in two different environments: regular Internet communications and traffic captures on sinkhole servers. For this purpose, we constructed two datasets: the TCP-Int dataset with public Internet traffic captures from the Stratosphere Project [11], and the TCP-Sink dataset with traces from a private sinkhole server gathered by INCIBE.

We have tested four supervised ML algorithms, namely DT, k-NN, SVM and NB, on the two datasets using an approach independent from the three possible network architectures used by botnets.

The TCP-Int dataset contained traces from three botnets: Kelihos, Miuref and Sality, aside from non-malicious data. We used the F1 score to evaluate the capability of the algorithms to identify the four classes. Both DT and k-NN showed a stable performance among the classes, and DT achieved the best mean F1 score (0.99). However, the performance of SVM and NB varied significantly between classes: the worst F1 score of SVM was 0.64 for the Sality class while the best score was 0.86 for the Kelihos class, and the best F1 score of NB was 0.49 for the Normal class while the best score was 0.75 for the Kelihos class. It is noticeable that the Kelihos class was the best identified by the four classifiers, revealing that its characteristics are more distinguishable from the other classes.

The TCP-Sink dataset contained Conficker botnet traces and normal traces from automatic Internet programs such as web crawlers. The four algorithms

were evaluated on this dataset using the AUC score, where the highest score was achieved by DT (0.99) and the lowest score was obtained by NB (0.88). Since the lowest score is relatively high, this means that the automatic non-malicious traffic in a sinkhole is easily identifiable.

On the basis of our results, we recommend DT for building botnet detectors on TCP traffic. For future works, we will study for an improved feature selection among the ones used in this work. The benefit from this is building lighter models that may run faster while maintaining the accuracy. Moreover, another optimization to consider consist on using assembled models from the ones used in this work, such as Random Forest or Boosted Trees from Decision Tree. Finally, our future work includes working with Deep Learning (DL) classifiers which can be trained with the same set of features of this work [28]. Besides, the text comprehension of DL Neural Networks could let us build featureless classifiers, i.e., classifiers than do not need a precalculation of features form the datasets, but can directly work with the data in text format.

Acknowledgements

This work was supported by the framework agreement between the University of León and INCIBE (Spanish National Cybersecurity Institute) under Addendum 01.

References

1. Martínez, J., Iglesias, C. and García-Nieto, P.: Machine Learning Techniques Applied to Cybersecurity. International Journal of Machine Learning and Cybernetics, 1–14 (2019)
2. Silva, S. S., Silva, R. M., Pinto, R. C., and Salles, R. M.: Botnets: A survey. Computer Networks, 57(2), 378–403 (2013)
3. Boshmaf, Y., Muslukhov, I., Beznosov, K. and Ripeanu, M.: Design and analysis of a social botnet. Computer Networks, 57(2), 556–578 (2013)
4. Bujlow, T., Carela-Español, V. and Barlet-Ros, P.: Independent comparison of popular DPI tools for traffic classification. Computer Networks, 76, 75–89 (2015)
5. Kirubavathi G. and Anitha, R.: Botnet detection via mining of traffic flow characteristics. Computers & Electrical Engineering, 50, 91–101 (2016)
6. Kim, H., Choi, S. S., and Song, J.: A methodology for multipurpose DNS Sinkhole analyzing double bounce emails. International Conference on Neural Information Processing, 609–616 (2013)
7. Fetzer, C., Felber, P., Rivire, ., Schiavoni, V., and Sutra, P: Unicrawl: A practical geographically distributed web crawler. International Conference on Cloud Computing, 389–396 (2015)
8. Sangkatsanee, P., Wattanapongsakorn, N. and Charnsripinyo, C.: Practical real-time intrusion detection using machine learning approaches. Computer Communications, 34(18), 2227–2235 (2011)
9. Kim, H., Claffy, K.C., Fomenkov, M., Barman, D., Faloutsos, M. and Lee, K.: Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. Proceedings of the 2008 ACM CoNEXT Conference, 11:1–11:12 (2008)

10. Doshi, R., Apthorpe, N., and Feamster, N.: Machine Learning DDoS Detection for Consumer Internet of Things Devices. IEEE Security and Privacy Workshops, 29–35 (2018)
11. García, S., Grill, M., Stiborek, J. and Zunino, A.: An empirical comparison of botnet detection methods. Computers & Security, 45, 100–123 (2014)
12. Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J. and Hakimian, P.: Detecting P2P botnets through network behavior analysis and machine learning. 2011 Ninth Annual International Conference on Privacy, Security and Trust, 174–180 (2011)
13. Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A. and Garant, D.: Botnet detection based on traffic behavior analysis and flow intervals. Computers & Security, 39, 2–16 (2013)
14. Buntine, W. and Niblett, T.: A further comparison of splitting rules for decision-tree induction. Machine Learning, 8, 75–85, (1992)
15. Friedman, J. H.: Lazy Decision Trees. Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1, 717–724 (1996)
16. Dong, W., Moses, C. and Li, K.: Efficient K-nearest Neighbor Graph Construction for Generic Similarity Measures. Proceedings of the 20th International Conference on World Wide Web, 577–586 (2011)
17. Cherkassky, V. and Ma, Y.: Practical selection of SVM parameters and noise estimation for SVM regression. Neural Networks, 17(1), 113 - 126 (2004)
18. Al Nabki, M.W., Fidalgo, E., Alegre, E. and de Paz, I.: Classifying illegal activities on TOR network based on web textual contents. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, 1, 35–43 (2017)
19. Fidalgo, E., Alegre, E., González-Castro, V. and Fernández-Robles, L.: Compass radius estimation for improved image classification using Edge-SIFT. Neurocomputing, 197, 119–135 (2016)
20. Fidalgo, E., Alegre, E., González-Castro, V. and Fernández-Robles, L.: Illegal activity categorisation in DarkNet based on image classification using CREIC method. International Joint Conference SOCO17-CISIS17-ICEUTE17, 600–609 (2017)
21. Fidalgo, E., Alegre, E., González-Castro, V. and Fernández-Robles, L.: Boosting image classification through semantic attention filtering strategies. Pattern Recognition Letters, 112, 176–183 (2018)
22. Schneider, K.: A Comparison of Event Models for Naive Bayes Anti-spam e-Mail Filtering. Proceedings of the Tenth Conference on European Chapter of the Association for Computational Linguistics, 1, 307–314 (2003)
23. Xu, S.: Bayesian Nave Bayes classifiers to text classification. Journal of Information Science, 44(1), 48–59 (2018)
24. Ren, J., Lee, S. D., Chen, X., Kao B., Cheng, R. and Cheung, D.: Naive Bayes Classification of Uncertain Data. 2009 Ninth IEEE International Conference on Data Mining, 944–949 (2009)
25. Sasaki, Y.: The truth of the F-measure. Teach Tutor mater, 1(5), 1–5, (2007)
26. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874, (2006)
27. Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., and Lin, C. J.: LIBLINEAR: A library for large linear classification. Journal of machine learning research, 9, 1871–1874 (2008)
28. van Roosmalen, J., Vranken, H. and van Eekelen, M.: Applying Deep Learning on Packet Flows for Botnet Detection. Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 1629–1636 (2018)