Machine Learning Fingerprinting Methods in Cyber Security Domain: Which one to Use?

Martin Laštovička*†, Antonín Dufka†, Jana Komárková*†
*Masaryk University, Institute of Computer Science, Brno, Czech Republic
†Masaryk University, Faculty of Informatics, Brno, Czech Republic
Email: {lastovicka|komarkova}@ics.muni.cz, xdufka1@fi.muni.cz

***Abstract*—Identification of a communicating device operating system is a fundamental part of network situational awareness. However, current networks are large and change often which implies the need for a system that will be able to continuously monitor the network and handle changes in identified operating systems. The aim of this paper is to compare machine learning methods performance for OS fingerprinting on real-world data in the terms of processing time, memory requirements, and performance measures of accuracy, precision, and recall.**

***Index Terms*—Machine Learning, OS Fingerprinting, IPFIX, Cybersecurity**

I. Introduction

The knowledge and understanding of the current situation in a network are required to protect the network against threats effectively [1]. Building cyber situational awareness (CSA) is a multi-stage process starting with network asset enumeration and comprehension of the systems in use. However, it is not enough to achieve this goal only once; we need to maintain in throughout time and update it with the changes that inevitably arise in each network.

The first step in CSA asset enumeration is determining the operating system of each active device. This task is particularly challenging in a vast dynamic network where devices connect and disconnect freely. In such environment, an active probing of devices is useless as devices behind an IP address change every few minutes at irregular intervals. Hence, the OS identification must work passively on captured network traces.

To monitor large networks, network flow [2] is currently the most preferred technology. It reduces the monitoring to the packet headers which are aggregated into the final flows. This approach enables monitoring of backbone networks, but the reduced visibility may pose a challenge for established OS fingerprinting methods as the required data might be unavailable. Matousek et al. [3] adapted the OS fingerprinting to the notion of network flow technology and Lastovicka et al. [4] showed the flow-based fingerprinting to be usable in large dynamic networks.

Passive OS fingerprinting methods usually rely on predefined fingerprint database. Such database is filled by fingerprints created manually by experts, based on statistical analysis of traces, or by collecting fingerprints from community volunteers (e.g., p0f). Each of the methods maintainability suffers as new systems are introduced much faster than the databases are updated. To solve this issue, machine learning algorithms were introduced [5] and further explored in the large-scale scenario by Richardson et al. [6].

When developing the machine learning methods, the authors usually focus on measures such as accuracy, precision, recall and test their methods on data from small controlled networks. However, the real network traffic is different in the number and diversity of operating systems, and the data volume is much higher. These network properties make many of the methods to remain prototypes unusable in practical deployment.

In this paper, we test the usability of four selected machine learning algorithms for OS fingerprinting of devices in a large network. We have collected network flows from a week of university wireless network traffic and paired them with operating system ground truth from DHCP and Radius servers' logs. On this dataset, we tested all methods focusing on their time and memory requirements while processing a large amount of data. We then combine this results with traditional accuracy metrics to select the method best suited for continuous network monitoring and OS fingerprinting.

The rest of this paper is structured as follows. Section II describes the selected ML methods together with the experiment settings and parameter optimisation. In Section III we describe the collected dataset and its usage in ML algorithms and Section IV presents the results of our measurements. We finish the paper with our lessons learned and an overview of similar work in Sections V and VI, followed by a conclusion.

II. Methodology

Our OS fingerprinting methodology is based on identifying a specific combination of TCP/IP packets parameters settings, whose default values are dependent solely on the OS kernel and typically differ among kernel implementations. To work with network flows, we chose three parameters that can be transformed from single packet observation to flow while the parameters are still sufficiently distinguishing for OS fingerprinting. The selected features are:

1) IP Time to Live (TTL) – Field specified by Internet protocol, whose value determines at most how many nodes can process the packet, before it shall be dropped. The initial value is set by the OS of the transmitting side, so it can be used as a factor in OS identification.
2) TCP Window Size – TCP Window Size is a TCP field specifies the number of bytes the receiver is currently

willing to receive, and its initial value is dependent on the OS settings.

3) Size of the initial SYN packet – The first packet in a TCP connection is a TCP SYN packet, which does not transfer any actual user-data, but is a necessary part of establishment of TCP connection. Its size varies among different operating systems, so we use this value as another source of information for the classification.

We avoided using parameters that are often used but which can depend on transmitted data (e.g., checksum, destination port) or network properties (e.g., MSS) [6] as they can influence the fingerprinting by other characteristics than OS kernel.

A. Machine Learning Methods

Based on the survey on machine learning methods for intrusion detection [7], we have selected four supervised classification algorithms with perspective time complexities which could be suitable for processing a large amount of network data.

1) Naïve Bayes: Naïve Bayes [8] is a classifier based on Bayes' theorem with an assumption of stochastic independence of given classes. The assumption is usually unrealistic in real-world usage. However, as shown in [8], it yields good results in many cases.

Fitting phase consists of analysing the training data and estimating posterior probabilities for individual classes, which are required for the Bayes formula. When trying to classify an unknown sample, the Naïve Bayes classifier uses previously estimated prior probabilities to compute probability, that the sample belongs to a given class. The final class is chosen as the one with the highest probability.

2) Decision Tree: A decision tree classifier [9] is based on recursive partitioning of the feature space and forming a tree-like structure, which holds the rules for deciding to which partition given sample belongs.

The fitting algorithm builds this tree-like structure. Every training sample corresponds to a certain point in feature space. The algorithm tries to find a hyperplane separating these samples into two half-spaces in such a way, that the information gain of this split is maximised. This step is recursively applied to the half-spaces, until all training samples are correctly classified, or the tree reaches its recursion limit.

The algorithm has to traverse the tree from root to leaf in order to classify a sample. For each inner node, the algorithm inspects a decision rule contained in the node and based on this rule decides which node will be the next. Once the algorithm reaches a leaf node, it finishes and reports the class found in the final node.

3) k-Nearest Neighbors: k-nearest neighbors (k-NN) [10] is a clustering classifier. The principle behind this method is to find k closest training samples and based on their classes predict one of the unknown samples. k-NN is an example of a non-generalizing method, which means that the classifier has to keep all the training data and that impacts its memory requirements. Parameter k determines how many nearest neighbours are going to be searched for during the classification phase. This parameter can be subjected to hyper-parameter optimisation to find its optimal value for the data.

The distance of elements is computed concerning the selected metric. Metric corresponds to a formula, which given two points in vector space returns a non-negative number (their distance). Commonly used metrics are Euclidean distance and Manhattan distance [10].

4) Support Vector Machine: Support vector machine (SVM) [11] is a binary classifier but can also be extended for multi-class classification. To do so, we use "One-against-all" method which trains one classifier per class to distinguish the class against the rest of the classes. Each of those classifiers is used during classification, and the resulting class is the one, whose classifier had the highest confidence score.

Training algorithm finds a hyperplane, that is dividing the training samples into two groups, with respect to their classes. The hyperplane is selected so that it maximises its minimal distance from the samples. By utilising kernel methods, the SVM can fit more complex patterns, than just a linearly separable data. It allows to map an input vector into higher dimensional spaces, and the hyperplane in those spaces can represent a more complex pattern when projected back into the original feature space. When classifying an unknown sample, trained SVM classifier finds out on which side of the boundary is the given sample and returns corresponding class.

Parameter C affects the penalty for wrongly assigned training sample. It affects the bias-variance trade-off. The higher the value of C, the more significant the penalty will be, which will result in overfitting. On the other hand, a too low value of C causes the boundary to be very simple which results in low accuracy.

B. Experiment Settings

The machine used for all the measurements has Intel® Core$^{\text{TM}}$ i5-4670 CPU and 16GB DDR3 RAM. In this section, we describe the used software and experiment setup.

1) Scikit-learn: Scikit-learn [12] is a popular machine learning library for Python, which provides consistent high-level API for efficient machine learning algorithms. All the selected methods use implementation from the library.

2) Data Processing Pipeline: Network flows from the dataset (described in detail in Section III) were stored in CSV file. Each record contained four values: OS, TCP window size, TCP syn size, and TCP TTL. These records were loaded into Pandas DataFrame, which is a memory-efficient tabular data storage suitable for Scikit-learn algorithms. Dataframe was split into labels (OS) and features (the other three values) and in this form passed to the classifiers.

3) Measurement: The measurement focused on both time and memory requirements of the algorithms. Time and memory were measured in separate runs, so their measurements did not interfere.

The execution time of the algorithm was measured using python $time()$ function. After training data and all python modules were loaded, the current timestamp was captured, and then the algorithm was started. When it finished, the starting time was subtracted from the current time, and this value was stored as a measurement of the time required to execute given algorithm on the specific dataset.

Memory measurement was based on querying operating system on how much memory is being used by the process during its execution. Like in time measurement, when everything necessary was prepared, OS was queried for how much memory is currently being used by the process, so that we can subtract this value and get just the memory used by the algorithm. During execution of the algorithm, the OS was periodically queried, and when it fished, the initial memory was subtracted from the maximal measured value, and the resulting value was stored.

Since both time and memory measurements are affected by minor deviations, most of the measurements were repeated 20 times, and the results were averaged. The measurements were performed with fewer repetitions when the number of samples used for measuring exceeded one million.

4) Learning and Hyperparameter Optimisation: Parametric classifiers had their parameter values chosen from individual sets of perspective values. Corresponding classifiers were then trained on a validation set with 10^6 samples for every combination of these parameter values. The combination, which yielded the best values of accuracy, recall and precision, was used in the experiment.

For the optimisation of *k-nearest neighbours* methods, parameters of the number of neighbours k, the weight of neighbours, and distance of neighbors were selected. The available values settings for k is an integer from the range $[0, 15]$, weight could be uniform or inversion of distance, and the distance metric could be Euclidean or Manhattan. Most of these combinations yielded similar accuracy, but the most promising results were obtained by selecting the value $k = 3$ with uniform weight and Euclidean metric.

Support vector machine has two parameters for optimization, penalty parameter C and kernel type. C was chosen from the range $[0; 10]$ with step of 0.03. Possible values for kernel can be chosen as *radial basis function* (also known as Gaussian), or *linear*. With the increasing value of C parameter, the accuracy grows but the recall decreases. The value $C = 1$ and "radial basis function" kernel was chosen as a compromise between precision and recall.

Decision tree and *Naïve Bayes* methods were run in the Scikit-learn default settings.

III. Dataset

Data for our experiments were collected using extended network flows. Our monitoring probes were located at the backbone links connecting the university to the Internet. For OS fingerprinting, we filtered the flows so that only flows with the source IP address from our wireless subnets (university Eduroam network) remained. Following this procedure, we limited the dataset size (traffic from the outside is irrelevant for fingerprinting of devices in our network) and focused only on the interesting samples of a real dynamic network.

The network flows were extended by fields necessary for OS fingerprint. TCP SYN size, TTL, and TCP win size were measured from the first packet of TCP connection (and hence first packet of network flow). We round the value of TTL to the next higher power of two to eliminate the influence of observation point location [13]. These values were assigned for the whole flow and exported using IPFIX protocol [14].

The ground truth values for OS fingerprinting were established by pairing each flow with corresponding wi-fi session. In our network, each wireless connection is authenticated, and we can access logs from DHCP and Radius servers. We mapped device names from these logs to current operating systems. In many modern devices, changing the device name is very hard for the user or even disallowed and the device manufacturers usually use an easily distinguishable default name (e.g., *android-<ID>*, *<user>-iPhone*, *Windows-Phone*). Hence, we consider this approach reasonably accurate as we want to keep the network open for any device, which results in a more diverse dataset.

We collected the data during the first week of May 2017 and it contains traffic traces of 25 642 unique devices (i.e., unique MAC addresses) from 21 746 unique user accounts and they produced 79 087 345 extended network flows. The whole anonymised dataset is publicly available on Github[1].

A. Dataset Modifications for Testing

The OS fingerprinting methods rely on the processing of TCP/IP parameters of each flow. As the dataset covers real traffic traces, the parameters naturally cannot be present in a significant portion of the flows (i.e., UDP, ICMP traffic). Similarly, some TCP flows do not contain parameters of the initial packet of the connection as the communication can be split into multiple network flows. Such flows are then unusable for selected OS fingerprinting methods and were removed from the dataset before testing. Following this procedure, we reduced the dataset to 13 660 089 network flows, where each contains all parameters needed and has ground truth value established. Then, we split the dataset into three smaller parts – training, validation, and test dataset.

The test dataset is the largest, containing 80 % of the samples. The remaining 20 % was evenly split between the two other sets. The splitting was done cautiously so that the individual parts preserve the same ratio of class distribution. Explicitly, the class distribution of the original set was computed. This value was used afterwards to determine the number of elements in each class of the new dataset. Then the individual records from the original dataset were sequentially inspected, and if there was still space in the new dataset for an element of corresponding class, it was added. If not, it was added to another dataset, that could be used for further splitting without having overlapping elements.

At the end of the data preparation, we had test dataset containing over 10 million samples, validation dataset with over 1 million samples and training dataset, again, with over 1 million samples, all of which had the same class distribution.

[1]https://github.com/CSIRT-MU/PassiveOSFingerprint/tree/master/Dataset

IV. Results

In this section, we present results of our performance measurements concerning the time and memory needed for training and classification.

A. Time

The time requirements on selected methods can be divided into two areas, the training time and classification time. Decision tree and Naïve Bayes achieved by far the best results. Their training time on 1 million samples was less than 1 second. Fitting of SVM classifier requires more complex computations which is reflected in results – with 1 million samples the time almost reached 5 minutes. k-nearest neighbours classifier was the slowest one to fit. It needed almost 15 minutes for fitting 1 million samples and continued to grow rapidly with additional samples.

The more important measure for practical use is the time it takes for a fitted classifier to classify a large number of samples. The complexity of the fitted classifier (how many samples it was fit on) did not profoundly affect classification times of Decision Tree, nor Naïve Bayes. On the other hand, its impact was significant on both SVM and k-nearest neighbours. Classification of 1 million samples by classifier fitted on one million samples was slower by $15.8\,\%$ for decision tree, by $0.9\,\%$ for Naïve Bayes, by $8\,668\,\%$ for SVM, and by $14\,288\,\%$ for k-NN than of the same classifier trained on 10 000 samples. Detailed view of this dependency can be found in Table I. These results show that SVM and k-NN methods are prone to building over-complicated models and their training phase should be treated carefully.

B. Memory

Similarly to the time, we measured memory consumption during fitting and then during classification. SVM classifier required the most memory for training, on 1 million samples it needed $429\,\mathrm{MB}$. The other three classifiers ended up with requirements on 1 million samples around $40\,\mathrm{MB}$ of used memory. The complexity of the classifier did not significantly affect memory requirements of classification algorithms; the most significant difference was of Naïve Bayes, which got an increase of 33 % between the classifier trained on 100 samples and the one trained on 100 000 samples.

Overall, the memory consumption depended mainly on the number of classified samples. Fig. 1 shows the amount of memory used while classifying a certain number of samples by classifiers trained on 100 000 samples. These measurements show just the additional memory used by classification algorithm, the memory needed for storing the trained classifier is a constant overhead, which can be neglected in comparison of memory dependency on the number of classified samples.

C. Accuracy, Precision, Recall

Accuracy, Precision, and Recall were calculated as an average value of the specific measure predictions over the classification dataset of classifier fitted on 100 000 training samples. Precision and Recall of individual classes were combined using the macro-averaging method according to Sokolova et al. [15].

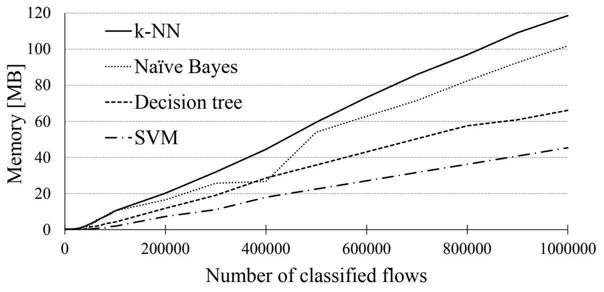

Fig. 1. Memory consumption of methods trained on 100 000 samples

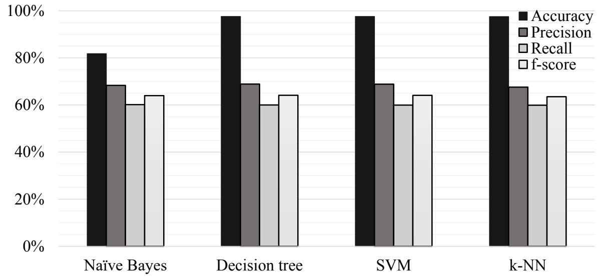

Fig. 2. Performance measures

With the increasing number of training samples, classifiers were able to increase accuracy, though not significantly. After a certain point k-NN, decision tree and SVM stabilised on 0.975, while Naïve Bayes on 0.818. Detailed values are listed in Table II and their visual comparison is depicted in Fig. 2.

The other performance measures show only minor differences between the methods. Unlike accuracy, precision and recall are not influenced by a large number of true negatives (which are typical for multi-class classification) and point out inefficiency in the identification of specific classes. The dataset contained many samples that were from different classes, yet had the same values of features. Therefore classifiers could not correctly classify many of such samples. That is apparent from recall which values ranged around 60 % for each method.

V. Lessons Learned and Discussion

When deciding which ML method to use, the goal is to pick the one, that can perform well on given data (has good accuracy, precision and recall) and requires a reasonable amount of resources to classify samples.

Memory requirements were not exceptionally large by any algorithm for fitting, nor classification. Modern computers usually have way more resources than were needed; therefore these criteria should not be the primary factor in deciding which method to use.

SVM, k-NN and decision tree had satisfiable accuracy and similar values of precision and recall, so if we were to consider

TABLE I
CLASSIFICATION TIME DEPENDENCY ON THE CLASSIFIER COMPLEXITY (IN SECONDS)

Training dataset size	10k			100k			1M		
Classified samples	10k	100k	1M	10k	100k	1M	10k	100k	1M
Naïve Bayes	0.0012	0.0111	0.2299	0.0012	0.0112	0.2304	0.0012	0.0112	0.2319
Decision tree	0.0004	0.0042	0.0759	0.0005	0.0049	0.0828	0.0005	0.0051	0.0876
SVM	0.3745	3.7790	37.976	3.4075	34.463	345.04	33.759	330.31	3330.0
k-NN	0.2981	3.0093	30.392	2.5196	25.730	252.31	42.652	417.67	4372.9

TABLE II
MULTICLASS CLASSIFIER PERFORMANCE MEASURES

	Accuracy	Precision	Recall	F-score
Naïve Bayes	0.8176	0.6830	0.6017	0.6398
Decision tree	0.9757	0.6887	0.5998	0.6412
SVM	0.9756	0.6886	0.5994	0.6409
k-NN	0.9755	0.6760	0.5990	0.6352

just these criteria, these methods would be almost equivalent. Thus the main deciding point will be their classification time.

In case we had a small amount of fine picked training samples, the SVM and k-NN could be used well. However, those methods become quite time-consuming as the number of training samples rises. Classification time of decision tree classifier, on the other hand, is not significantly affected by the number of training samples and therefore appears to be the best choice (of the investigated classifiers) for OS fingerprinting.

VI. RELATED WORK

The research focused on OS fingerprinting is mostly focused on developing new methods with high accuracy. As far as we are aware no work focuses on performance benchmarking of OS fingerprinting methods on large datasets. However, the related field of application fingerprinting has several works that consider not only accuracy but also computational performance. In this section we present relevant papers on OS and application fingerprinting.

Lippmann et al. [13] propose a new approach for passive OS fingerprinting using TCP/IP packet header information, such as TCP window size, IP time to live, TCP max segment size, IP dont fragment, TCP selective acknowledgements options flag, packet size, and SYN and SYN-ACK packet flags. They tested several machine learning algorithms including support vector machines, nearest neighbours, binary tree classifier and multi-level perceptron. They achieve 10 % error rate without rejection when considering nine classes of OS. They conclude that the low error rate can be achieved only by decreasing the total number of classes.

Al-Shehari and Shahzad [16] use combination of pattern matching methods and machine learning to detect OS from network traffic. They analyse TCP/IP headers such as time to live, window size, don't fragment bit, and TCP options/flags. They correlate the packets from the same TCP/IP session mainly the three-way handshake with session termination. Moreover, their method can be used even with SSL communication. For evaluation of their approach, only the accuracy is considered, which is around 90 %.

Este et al. [17] apply SVM on the classification of L7 protocol based on flow data. They evaluate their approach on datasets labelled based on port number pre-classifier, achieving accuracy over 90 %. They theoretically discuss the learning and classification time complexity.

Li et al. [18] uses flow data for application fingerprinting. They distinguish the traffic into classes based on the type of communication, such as mail, p2p, VoIP, games, database, etc. They try several approaches, ranging from machine learning to packet inspection, and evaluate their accuracy as well as performance benchmarks. They compare the stability of the approach, using the trained method on different time windows in the same network and the same time window in different networks. The packet inspection based algorithms prove not very robust in time.

Williams et al. [19] compare several machine learning techniques for L7 protocol identification from flows. They compare nave Bayes, C4.5 decision tree, Bayesian network and nave Bayes tree algorithms. They focus on exploring the effect of feature reduction on accuracy a computation and they conclude that the feature set can be significantly reduced with a minimal decrease in classification accuracy.

Soysal et al [20] investigates three flow-based traffic classification methods based on supervised machine learning, namely Bayesian networks, decision trees and multilayer perceptrons. The methods are evaluated based on performance metrics of correctness and computational cost (model build time and classification rate). They distinguish six types of network traffic: HTTP, Akamai, FTP, DNS, SMTP, and P2P. They explore the effects of training dataset size on the accuracy of each method.

Lee at al. [21] introduces internet traffic classification benchmark tool, NeTraMark. Its purpose is to enable easy objective comparison of various machine learning approaches to application classification. The tool automatically compares per-trace accuracy (overall accuracy), per-application accuracy (precision, recall, and f-measure), and computational performance (learning time and classification time). The benchmark tool follows six design guidelines: comparability, reproducibility, efficiency, extensibility, synergy, and flexibility/ease-of-use. The tool already includes eleven state-of-the-art traffic classifiers and several projects consisting of datasets and benchmarks for those datasets.

Nguyen and Armitage [22] survey general machine learning usage for IP traffic classification. They review the most significant works published on traffic classification based on machine learning. They discuss the key requirements for the employment of machine learning based traffic classifiers in operational IP networks and summarise to which extent are the requirements met in each work.

VII. Conclusion

In this work, we have selected four machine learning algorithms and tested their usability for OS fingerprinting of real-world network flow data. We have collected a large dataset of traces from a wireless network, where users can bring and connect an arbitrary device with generally no regulations. We have published the anonymised version of the dataset for public use as we think such data is quite uncommon in the research community.

The goal of the methods performance testing was to find out, which one is the best for deployment and processing of data from a high-speed network. The most critical parameter is the ability to process a batch of data before next batch arrives. We measured classification time of each method with different settings of training data and the Naïve Bayes and Decision tree methods outperformed the others (SVM, k-NN) by several orders of magnitude. In our next measurement, we focused on the memory consumption of each method, which showed considerable differences between the methods, but each of them could work within the memory of nowadays cheap mobile phones. Hence, we conclude the memory is not a limiting factor for any of the methods. Finally, the popular measures of accuracy, precision, recall, and f-score were calculated for each method. Our results show there are no significant differences except for slightly low accuracy of Naïve Bayes. Otherwise, the methods are comparable to each other.

Decision tree method decidedly outperformed other methods and proved itself to be the most suitable for flow-based OS fingerprinting. Accuracy measures and memory consumption are comparable to others, but training and classification times make the difference.

Our next research in the field of OS fingerprinting will tackle the usage of new network protocols which introduce new features and shift the meaning of the current ones. Specifically, deployment of IPv6 protocol changes the semantics of Time to Live (called Hop Limit) and packet Total Length (renamed to Payload Length), and hence, those features need to be investigated whether they are still usable for OS fingerprinting. Similarly, QUIC protocol by Google is gaining more traffic share nowadays, and features of this protocol need to be discovered as it moves the traffic to UDP connections.

Acknowledgement

This research was supported by the Security Research Programme of the Czech Republic 2015 - 2020 (BV III / 1 VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20172020070 Research of Tools for Cyber Situation Awareness and Decision Support of CSIRT Teams in the Protection of Critical Infrastructure.

Martin Laštovička is Brno Ph.D. Talent Scholarship Holder – Funded by the Brno City Municipality.

References

[1] A. Kott, C. Wang, and R. F. Erbacher, *Cyber Defense and Situational Awareness*. Springer, 2014, ISBN: 978-3-319-11390-6.

[2] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, Fourthquarter 2014.

[3] P. Matoušek, O. Ryšavỳ, M. Grégr, and M. Vymlátil, "Towards identification of operating systems from the internet traffic: Ipfix monitoring with fingerprinting and clustering," in *Data Communication Networking (DCNET), 2014 5th International Conference on*. IEEE, 2014, pp. 1–7.

[4] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive OS Fingerprinting Methods in the Jungle of Wireless Networks," in *Network Operations and Management Symposium (NOMS), 2018 IEEE*.

[5] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, "Fig: Automatic fingerprint generation," *Department of Electrical and Computing Engineering*, p. 27, 2007.

[6] D. W. Richardson, S. D. Gribble, and T. Kohno, "The limits of automatic os fingerprint generation," in *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*. ACM, 2010, pp. 24–34.

[7] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[8] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22. IBM, 2001, pp. 41–46.

[9] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[10] S. Marsland, *Machine learning: an algorithmic perspective*. CRC press, 2015.

[11] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[13] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, "Passive operating system identification from tcp/ip packet headers," in *Workshop on Data Mining for Computer Security*, 2003, p. 40.

[14] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," Internet Engineering Task Force, 2013.

[15] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[16] T. Al-Shehari and F. Shahzad, "Improving operating system fingerprinting using machine learning techniques," *International Journal of Computer Theory and Engineering*, vol. 6, no. 1, p. 57, 2014.

[17] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for tcp traffic classification," *Computer Networks*, vol. 53, no. 14, pp. 2476–2490, 2009.

[18] W. Li, M. Canini, A. W. Moore, and R. Bolla, "Efficient application identification and the temporal and spatial stability of classification schema," *Computer Networks*, vol. 53, no. 6, pp. 790–809, 2009.

[19] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[20] M. Soysal and E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Performance Evaluation*, vol. 67, no. 6, pp. 451–467, 2010.

[21] S. Lee, H. Kim, D. Barman, S. Lee, C.-k. Kim, T. Kwon, and Y. Choi, "Netramark: a network traffic classification benchmark," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 22–30, 2011.

[22] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.