Doctoral Dissertation
Doctoral Program in Electrical, Electronics, and Communications Engineering
(32^{nd} cycle)

Machine Learning and Big Data Approaches for Automatic Web Traffic Monitoring

Andrea Morichetta
* * * * * *

Supervisor
Prof. Marco Mellia, Supervisor

Politecnico di Torino
January 9^{th}, 2019

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Andrea Morichetta
Turin, January 9^{th}, 2019

Summary

The analysis and monitoring of network traffic are of fundamental importance for numerous activities related to its management. These activities are involved in quality control of the service, support the planning and updates of the network based on traffic load, and contribute to the development of advanced security systems and the identification of malicious attacks. Therefore, approaches oriented to the processing of data represented by traffic traces are appropriate to understand the conditions and behavior of the network.

Particularly critical is the analysis of the Hypertext Transfer Protocol (HTTP) traffic. The last years have seen the proliferation of applications and services that rely on HTTP. The complexity of the Web is increased and, consequently, its analysis. What is more, cyber-criminals in the years have deployed more sophisticated and stealthy ways to generate and spread their malicious content through HTTP traffic. In this direction, many researchers and companies are focusing on data analysis and machine learning techniques. Many solutions have been developed, but often pinpointing just particular problems.

The thesis, therefore, proposes to provide a generic methodology for monitoring HTTP traffic; in particular, it aims to identify new services, anomalies, and suspicious traffic, looking at URLs. The Uniform Resource Locator (URL) is a unique address for a particular web resource, such as an image, a video or a HyperText Markup Language (HTML) file and is characterized by two components: the hostname, or the name of the server that owns the object and the object path or the name (and path) of the object. We have seen that, in general, malicious URLs tend to have characteristics that make them visually different from benign ones. For example, malicious organizations tend to use artificially generated hostnames composed of random strings that do not contain common or easily memorable names, as is the practice for legitimate time organizations, to avoid detection by black-lists. In general, the URLs contain essential information about the related services, creating in that way structures that are identifiable.

In detail, the work of this thesis develops the idea mentioned above, addressing the following research questions. The first one is (i) how to automatically reduce the amount of traffic, creating meaningful groups, (ii) how to let the grouping technique being adaptive for different kinds of data, i.e., URLs, without a constant need to manually tune the parameters, (iii) how to scale up to a big data problem, (iv) how to check the occurrence

of new traffic and how to build a history of the previous collected information.

This thesis presents a self-tuning clustering solution, as grouping technique, called Iterative DBSCAN. It consists of iteratively run DBSCAN, a popular clustering algorithm, each time using a different value of the input parameters, in order to extract clusters that, after an evaluation, result to be well-shaped, according to quality metrics. To group URLs, I considered them as strings, sequences of characters using metrics that allow measuring the difference between two sequences.

Clustering execution is, however, a computationally demanding task, especially with complex distance functions. This thesis aims at untieing the distance computation, from the algorithm execution, to overcome this performance bottleneck. This approach, together with the gains of distributed platforms like Apache Spark or MapReduce, guarantees a faster execution of the algorithms, together with more flexibility in the choice of the clustering method.

In order to analyze the evolution of the traffic over time, this thesis presents the implementation of a self-learning methodology, where the system grows its knowledge, which is used in turn to automatically associate traffic to previously observed services, and identify new traffic generated by possibly suspicious applications. The whole system takes the name of LENTA - Longitudinal Exploration for Network Traffic Analysis.

The developed methodologies are essential tools for the network administrator, be it a corporate network or a provider. The operator will be able to generate clusters starting from the URLs contacted by the employees of the company (or ISP customers) and, starting from this aggregate view, identify the activities related to malicious behavior. Following this analysis, the administrator can apply filters on these unwanted contents within the network. These approaches ensure greater security against malicious attacks, for the network itself and for the hosts that make it up, without affecting the quality of the user's navigation. Furthermore, the proposed methodologies let the analysts easily observe changes over time in network traffic, identify new services, and unexpected activities. The work is applied over HTTP and HTTPS data. The former case makes use of passive traces, while the latter is the outcome of data collected using a proxy installed on users' devices. In particular, this second scenario requires specific care concerning privacy and shows the potential of the proposed techniques in an enterprise context.

Acknowledgements

This thesis is the result of the help and support of many people. The path that brought me here has been characterized by more and less easy moments. I thank Marco for having always trusted my potential and helped whenever it was needed. I thank my colleagues, beautiful people, whom I learned about over time and with whom I shared unforgettable moments inside and outside the office. From Hassan and Enrico, with whom I shared the first months, to Martino and Luca, who supported and helped me until the last moment. Thanks to Michele, Eros, Dena, and Nicola, who have always given me a smile in these last months. Thanks to Azadeh, to Francesca. Thanks to Andrea for all the nights together, to Giuseppe, Greta, and Marco. Many thanks to Pedro, Matteo, Sofia, Micki, Mihai, Nikolas.

Thanks to all the people I met in Turin, with whom I felt at home. Thanks to the Flowers, for the good times together. Thanks to Margherita, to Leonardo, for all the conversations and for life lessons. Thanks to Tommaso, for the "maldite" dinners and the detox evenings. A huge hug to Matteo, we will never stop being there for each other. We've been through a lot together, and I'm looking forward to living as many.

A huge thank you to my family, which has never failed me to lack support, help, and understanding. Thanks, Ale, for having been there in the most decisive moments, you know. Thanks to my friends from Macerata, who I found again, stronger than before, and who never left. Finally, thanks to you, Julia, you have turned my life upside down over the past year. It has been incredible, and I hope it's the beginning of an even more amazing future. I am looking forward to everything.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The current decade has seen the growth of the Internet at an astonishing speed. The web has especially changed a lot in the last years. New services have risen, starting from social media to diverse streaming platforms. Contemporaneously, the number, habits, needs, and behavior of active users wholly changed. According to the last statistical analysis of We Are Social[1] and Hootsuite[2], published in October, 23^{rd}, 2019 [19], the number of users grew by more than 400 million, a 10 percent increase over the last year in total. Also according to We Are Social and Hootsuite [18], the average worldwide time spent on the Internet by users is almost 7 hours per day. Games, for example, are reportedly a considerable part of the Internet traffic, and they will likely increase their significance in the future, giving the new potential given by cloud platforms. In this context, the last in order of time the interesting scenario of game streaming provided by Google Stadia.[3] At the same time, the Internet is still not safe from malware and attackers. The new threats are becoming more targeted, and are challenging different fields, for example attempting on businesses and interfering with elections [1]. Furthermore, nowadays, Web traffic traces are studded with a substantial quantity of third-party services. They are often online tracking services or *trackers* [51]. They follow users' activities to create profiles, that they usually, but not only, sell to online advertisers. This practice is by now everyday nature for most of the users, and while it can be favorable, it also presents privacy and ethical concerns [70].

In this context, Hypertext Transfer Protocol (HTTP) is the de-facto standard application-layer protocol [63]. HTTP allows the browser to retrieve the hundreds of objects composing a page with a simple request-response mechanism. While the massive adoption of HTTP has simplified the structure of the protocol stack, the complexity

[1] https://wearesocial.com/

[2] https://hootsuite.com/

[3] https://stadia.google.com

of current developments raveled the analysis of web traffic, so that it is tough to understand which services are running on the network and what information they are carrying. Billions of objects are available on the web, each of them identified by a Uniform Resource Locator (URL). Static URLs directly point to an object, e.g., portions of a text or an image file like `http://acme.com/index.html`, but more and more frequently URLs encode queries that servers process to return a dynamic result. For instance, a Google search, a click on "like" buttons, or the images served by an advertisement platform are typical examples of dynamic URLs, e.g., `http://acme.com/s?key=like`.

Given the number of URLs that are retrieved to fulfill ordinary browsing activities, monitoring, and understanding the dynamics of the network is not an easy task. In a corporate scenario, the network analyst is interested in periodically processing the traffic to observe which services are accessed by terminals, to take informed actions then. This task requires to process a consistent amount of traffic so to guarantee the correlation and comparison between events that a too coarse analysis would miss. This scenario calls for the support of automatic tools to process, analyze, and extract useful information from the raw data, i.e., a big data solution.

Network Traffic Monitoring and Analysis (NTMA) Understanding how Internet services are operating and how users access them is critical to many applications. Network Traffic Monitoring and Analysis (NTMA) is central to that task. Applications range from providing a view on network traffic to the detection of anomalies and unknown attacks while feeding systems responsible for usage monitoring and accounting. They collect the historical data needed to support traffic engineering and troubleshooting, helping to plan the network evolution and identify the root cause of problems. It is correct to say that NTMA applications are a cornerstone to guarantee that the services supporting our daily lives are always available and operating as expected.

Traffic monitoring and analysis is a complicated task. The massive traffic volumes, the speed of transmission systems, the natural evolution of services and attacks, and the variety of data sources and methods to acquire measurements are just some of the challenges faced by NTMA applications. As the complexity of the network continues to increase, more observation points become available to researchers, potentially allowing heterogeneous data to be collected and evaluated. This trend makes it hard to design scalable and distributed applications and calls for efficient mechanisms for online analysis of large streams of measurements. More than that, as storage prices decrease, it becomes possible to create massive historical datasets for a more accurate retrospective analysis.

These challenges are precisely the characteristics associated with what, more recently, has become known as *big data*, i.e., situations in which the data *volume*, *velocity*, *veracity* and *variety* are the key challenges to allow the extraction of *value* from the data. Indeed, network traffic monitoring and analysis were one of the first examples of big data sources to emerge [49], and it poses big data challenges more than ever.

It is thus not a surprise that researchers are resorting to big data technologies

to support NTMA applications (e.g., [40, 47, 60, 77]). Distributed file systems – e.g., the Hadoop[4] Distributed File System (HDFS), big data platforms – e.g., Hadoop and Spark, and distributed machine learning and graph processing engines – e.g., MLlib and Apache Giraph, are some examples of technologies that are assisting applications to handle datasets that otherwise would be intractable. This scenario, certainly promising from a technological point of view, opens up new challenges. Undoubtedly, one of the most interesting is to combine in the field of NTMA big data approaches with advanced analysis mechanisms, including Machine Learning and Artificial Intelligence methodologies.

Machine Learning and Artificial Intelligence The availability of massive amounts of data and the necessity of fast reactions to events, together with the undeniable rise in popularity of Artificial Intelligence (AI), and in particular Machine Learning (ML) techniques, have concerned the network community in the last decade [59, 12]. The capability of addressing big data problems and automating processing measurements is appealing for multiple problems, from network security to Quality of Experience (QoE) monitoring and analysis [12].

When it comes to ML techniques and methodologies, the reference is often and more extensively pointed to supervised approaches. Supervised learning builds a model starting from the data, requiring these to be a priori categorized, i.e., labeled according to the ground truth. Ground truth is generally missing due to the structural complexity of the data, limits of human knowledge, and significant volumes that complicate the categorization process. This scenario is especially critical when it comes to network traffic, where researchers and practitioners have indeed to deal with small and outdated datasets [3].

Unsupervised techniques offer a solution to this lack of ground-truth since their goal is to analyze the structural properties of the data, based on some form of similarity among data instances. Different approaches are possible, depending on the overall goal (e.g., outlier detection, categorization, and others), and the different levels of complexity of the analysis. In any case, there is less need for ground truth.

Network Security and Network Characterization Many NTMA applications have been proposed for assisting cyber-security [42]. The most common objective is to detect security flaws, viruses, and malware to isolate infected machines and take countermeasures to minimize damages. Roughly speaking, there are two main approaches when searching for malicious network activity: (i) based on attack signatures; (ii) based on anomaly detection.

Signature-based methods build upon the idea that it is possible to define fingerprints for attacks. A monitoring solution inspects the source traffic/logs/events searching for

[4] http://hadoop.apache.org/

(i) known messages exchanged by viruses, malware, or other threats; or (ii) the typical communication patterns of the attacks – i.e., similar to behavioral traffic classification methods. Signature-based methods are efficient in blocking well-known attacks that are immutable or that mutate slowly. These methods, however, require attacks to be well-documented.

Methods based on anomaly detection [10, 26] build upon the assumption that attacks will change the behavior of the network. They build models to summarize the *normal* network behavior from measurements. They monitor live traffic, and they trigger alerts when the behavior of the network differs from the baseline. Anomaly detection methods are attractive since they allow the early detection of unknown threats (e.g., zero-day exploits). These methods, however, may not detect stealth attacks (i.e., false negatives), which are not sufficiently large to disturb the network. They sometimes suffer from large numbers of false positives too [3].

Thesis proposal My work proposes the use of unsupervised machine learning, i.e., clustering, to explore the network traffic and users' data, in particular, URLs, in order to extract patterns in an aggregated view and understand better what is happening in the network. Clustering can help to reduce the size of the problem from a hundred thousand single objects – the unique URLs – to few hundreds of clusters containing "similar" URLs. Notice that most URLs carried by a network do not derive from an intentional user action (e.g., the click of a link on a page), but are instead due to applications fetching objects (e.g., elements in a web page, or system component for a web-app) [74]. These latter groups often have a regular syntax, which makes them strictly different, but similar in the format. Designing a clustering solution for URLs requires ingenuity, given URLs are strings, for which the notion of similarity is not trivial to define.

The work described in this thesis aims at answering the following research question: *How to group and extract significant HTTP/S traffic patterns looking at the URLs lexical similarity with no other external information*? This question is rich and includes many research queries that I will address in Section 1.1. Several works tackled these topics before, and they are reported in Section 1.2.

The proposed solution leverages a novel density-based clustering algorithm for the grouping, using string-based distances. This algorithm enhances classic clustering methodologies by simplifying the parameter choice, a frequently cumbersome process. I call it Iterative DBSCAN, or IDBSCAN for short. When clustering URLs, it outperforms other off-the-shelf clustering algorithms in grouping URLs with a similar structure. The analyst can, at this point, examine URLs in each cluster to identify the service that generated them, i.e., giving them a possible label. Next, I design a self-learning approach that lets the system build *System knowledge*. This approach works comparing newly found clusters to those found in the past, so to automatically re-assign the same label if already known. In this way, the analyst will have to inspect only previously unseen clusters, while known traffic is automatically labeled. This process lets the analyst highlight changes and the birth of previously unseen traffic, building a longitudinal view.

The overall solution, IDBSCAN and the System Knowledge, takes the name of LENTA. I test LENTA on three real use cases, explained in Chapter 5. The first two consist of traffic collected through a passive probe that observes thousands of users in an ISP network. The first one considers one day of traffic, with the combination of labels for malware, the second one for three weeks. In the third case, the HTTPS gathering occurs through the use of a Man In The Middle (MITM) proxy to collect and process all traffic coming from single hosts. The first two cases consider all unique URLs generated by hosts in the monitored network, mimicking the case of the network analyst that is interested in observing what the network carries. Instead, traces collected by the proxy are processed considering all URLs generated by every single device, mimicking the case of the security analyst that has access to HTTPS traffic and is interested in each single device tracing. The results show (i) the ability to aggregate hundreds of thousands of URLs into few clusters, which are easy to investigate and associate to services or malicious activities, and (ii) the capability of identifying new traffic generated by previously unknown applications.

1.1 Research Questions

The main interrogation guiding this thesis, and reported in the introductory section, is the result of different and evolving questions that have arisen during the three years of Ph.D. I list, in the following, the questions that drew, step by step, a path in these years, and that I addressed throughout this thesis.

(i) *How to group similar malicious traffic?* This question starts together with the study of a particular class of malware that uses the so-called DGA (Domain Generation Algorithm) technique to elude static controls based on blacklists. The DGA technique generates pseudo-random domains starting from shared seeds (e.g., current date or Twitter trends). However, the URLs that result from this class of threats are often similar, and it is visually clear to find a match between them.

Given this brief introduction, it is possible to discuss the two main concerns that this question embeds. The first and foremost is how to express the similarity (or the dissimilarity) between strings. It is necessary to find a formal value that can express this notion. The second concern is about which technique to use. The objective is to group similar objects, i.e., URLs, of which we may have or not have previous knowledge about the type.

(ii) *What happens after grouping a collection of URLs and what the formed groups contain?* This question implies the analysis of the results in toto, and a metric - or an equivalent - for its evaluation. The current interrogation implicitly contains two questions: is this methodology effective in extracting knowledge about malicious traffic? Is it able to discover other attractive behaviors? The analysis focuses on answering these questions.

(iii) *How to automate the process, especially in a fully unsupervised scenario?* The considered techniques usually require a set of parameters in order to adapt to the data. Sometimes these parameters are not transparent or are strictly linked to the structure of the dataset to analyze and thus challenging to tune in case the scenario is entirely unsupervised. Therefore, there is the need to find ways to automate the parameter choice. This task also implies the investigation of methodologies to self-regulate the quality of the formed groups.

(iv) *How to address the scalability problem and speed up the extractions of meaningful groups?* Computing the dissimilarity between points is the main bottleneck for many relevant methodologies. Indeed, most of the solutions proposed in the literature, if they offer complete and not approximated solutions, they require the computation of all the pairwise distances between the points of the collection. The effect is a complexity of $O(N^2)$. There is, thus, the need to develop different solutions.

(v) *How can it be possible to analyze the evolution - in terms of update, birth, and disappearance - of those groups over time?* Analyzing one snapshot of traffic through the aggregation provided by grouping is undoubtedly useful. More intriguing is to develop a temporal analysis of those groups. The inspection over time helps many possible interpretations, from user behavior analysis to network security. This target requires the definition of a structure for the storage, control, maintenance, and update of the groups.

(vi) *What happens in different scenarios? Is the system generalizable?* This last question is the starting point to evaluate other contexts where it could be possible to apply the solution. It is necessary to cover the variety of different cases in the considered problem, as well as checking the robustness of the implemented solutions.

These questions have been addressed in different published works, evolutionarily. Questions (i) and (ii) are part of the first work, CLUE - Clustering for URL Exploration [57], where I introduce and apply density-based clustering for URLs grouping, focusing on malicious activity, and looking at lexical similarities. Question (iii) is part of LENTA - Longitudinal Exploration for Network Traffic Analysis [55]; I introduced IDBSCAN and the self-learning methodology there. The work presented in [23] offers an insight on the issue of scalability in density-based clustering of question (iv). Finally, the papers [54, 56] extensively handle questions (v) and (vi).

1.2 Related Approaches

The thesis proposes an approach for network traffic analysis at the application level, using unsupervised techniques in a continuous evolutionary way. Categorizing and classifying web data and traffic is a relevant topic of the last decade, for many diverse

applications. Together with the World Wide Web explosion, the need to understand, analyze, and categorize it to offer a more transparent structure rose rapidly. Clustering techniques have been immediately appealing, given their capability of working in an unsupervised fashion and offering a view of patterns and categories in the Web.

Several papers in the literature aim at identifying similar web pages or URLs. Each work is targeting different problems or ties to a specific application, with the design of custom techniques. The vast majority of works look for structural features that help in distinguishing different classes of websites to consequently group or classify them. Such features can refer to (i) to the URL of a web page, here intended as a sequence of characters; or (ii) to the payload of the page, consisting of its layout, formatting, and syntactical properties.

A group of previous works aims at clustering web pages directly using the text they contain. Such an approach requires the complete retrieval of the page and typically expensive text-processing algorithms. A notable example of clustering applied to web content is [13]. The authors propose a methodology to quantify the syntactic similarity between generic text files through the computation of resemblance and containment features. They apply such technique to 30 M documents retrieved from the Web and run clustering algorithms on top. A similar and more recent approach is presented in [17], while [33] stresses the importance of algorithmic design to achieve high scalability of clustering algorithms.

In the context of web page clustering for specific applications, the authors of [20] apply clustering algorithms to disambiguate between people's names on the Web. They use a set of features coming both from the page content and from the URL. They split the URL into multiple components (e.g., domain name, path, parameters) and extract properties that have to be recombined together, making the whole process a precise but expensive technique.

More recently, Hussain et al. in [35] present a filtering system able to classify multilingual URLs, according to URL characteristics and web page metadata.

1.2.1 Network Traffic Analysis

Thanks to the complexity and richness of network traffic, the last decade has witnessed several research studies concerning the use of machine learning techniques to automatically extract information. The critical applications are traffic classification and anomaly detection. In both cases, the employment of clustering techniques is of great interest.

Authors of [30, 29] addressed the task of botnet detection. The former uses a two-step clustering of communication flows, first coarsely grouping them considering a contraction of the feature space, and then, for each group, computing a more refined cluster, considering all the features. The latter uses a hierarchical clustering technique to merge similar bags of bi-grams, extracted from messages collected from Internet Relay Chat

monitoring. Converse to our solution, they focus uniquely on a specific target, i.e., botnets, and they use different features and techniques for clustering.

Considering traffic classification, Erman et al. [21] use transport layer statistics and test clustering algorithms (namely, k-Means, DBSCAN, and AutoClass) over different labeled datasets. Authors of [43] use marked traffic flows and k-Means clustering for the classification of TCP flows. Wright et al. in [76] leveraged k-Means over Hidden Markov Models of Client-Server and Server-Client communications as an intermediate step toward the detection of application behavior over encrypted traffic. The goal is, again, traffic classification. In this thesis, the focus is on HTTP/HTTPS traffic and on extracting clusters looking at the structure of the URLs.

Some studies focused on text and string mining techniques, with the goal of clustering network traffic. In the field of network security, authors of [62] use a two-level clustering process, leveraging the single-linkage hierarchical algorithm to disclose similarities between malicious URL; Levenshtein distance, together with Jaccard Index, is used in the second clustering stage. They target malware signature building explicitly. In [38], semantic features of the URLs are used to target the same problem, using DBSCAN and Jaro-Wrinkler distance. Authors of [48] use the Levenshtein distance aiming at detecting phishing sites whose names are built using typical spelling mistakes. Gao et al. [25] use clustering techniques to identify spam campaigns on Facebook, looking at similarities in destination URLs. Other works target YouTube traffic [27], or P2P traffic [8], and use features extracted from TCP flows like round trip time, data exchanged, and other domain-specific metrics. The authors in [58] developed an IPFIX-based big-data lightweight methodology for traffic classification. The work uses unsupervised learning for word embedding on Apache Spark, receiving as input ”decorated flow summaries,” which are textual flow summaries augmented with information from DNS and DHCP logs.

All these works focus on a specific class of traffic, with a distinctive goal. Here, the aim is broadly exploring HTTP traffic, in general. This work focuses on URLs, which are strings, for which defining a distance and a clustering requires ingenuity. LENTA is a general methodology that examines URLs from HTTP traffic to group elements that look similar. This approach allows the analyst to quickly identify patterns, anomalies, and novelties in web traffic, services, and users' behaviors. The system proposed considers all HTTP traffic, and not just malicious URLs or URLs generated by malware during their activity.

1.2.2 Clustering

This thesis also proposes a particular clustering technique based on the use of DBSCAN as a baseline algorithm, called Iterative DBSCAN. Several different works before the one presented here tried to enhance DBSCAN performance and usability. OPTICS [6] and HDBSCAN [50] are the main ones. This thesis includes the evaluation of

these techniques, in Ch. 3, testing the considered methodologies on the datasets presented in Ch. 2. The results show how IDBSCAN outperforms the other techniques. Clustering is considered one of the most interesting unsupervised learning techniques, providing a data structure partition that can be the basis for further learning. In particular unsupervised techniques have provided the possibility to obtain various forms of clusters and to separate the noisy points from the final result. However, the sensitivity to the parameters and the inability to manage data sets with different densities has represented a limit. For these reasons, other methodologies started to emerge. OPTICS [6] from Ankerst et al. offers a method to inspect the structure of clusters obtained from the data – the reachability plot – that is used to extract groups looking at density valleys with a visual inspection - a not always applicable solution. Campello et al. proposed HDBSCAN [14, 50], which is built on top of DBSCAN and considers different radius values, combining the results to find the best clustering. However, it offers limited control of the quality of clusters. In our proposed solution, IDBSCAN, the clustering stage builds upon the classic DBSCAN. It makes use of silhouette to allow better results in terms of quality concerning other well-known and novel algorithms. Furthermore, the automation of the choice of cumbersome parameters reduces the effort needed by the analyst to configure and tune the system. The results are in Ch. 3 and they report the test of the considered methodologies on the datasets presented in Ch. 2.

1.2.3 Scalability Issue

From a system point of view, this work addresses the engineering and deployment of scalable clustering algorithms, in particular, for density-based algorithms. We adopt big data approaches for which ingenuity is required to parallelize the execution. In the system community several authors are working on scalable clustering solutions [16, 31, 46, 34]. They mostly take advantage of feature space partition, leveraging Spark, and the MapReduce paradigm. These approaches, however, are confined to the class of problems where Euclidean distance metrics can be used and provide approximated clustering. Recently, Lulli et al., in their work NG-DBSCAN [44], propose a methodology to overcome those limitations that consist in an approximated version of DBSCAN, based on a *vertex-centring* programming paradigm, built on the concept of graphs. It provides an increase in terms of performance and scalability at the cost of lower accuracy. Other works realize parallel optimizations of density-based algorithms, focusing on GPU computing capabilities [75], or the multiprocessing API OpenMP in conjunction with graph techniques [61].

Our approach follows a different angle and targets the parallelization of the distance matrix computation. This strategy stems from the fact that the estimate of string similarity is per se a resource-demanding job. Being URLs possibly very long strings, this poses severe scalability issues that are solved by providing a map-reduce solution. Once this algorithm extracts the distance matrix, IDBSCAN can run in a centralized manner, thus not facing any approximation, i.e., not losing information.The proposal

is to decouple the distance computation from the algorithm execution, allowing inexpensive experiments with different clustering algorithms and faster parameter tuning. Moreover, some clustering quality measures are based on the concept of cohesion and separation, thus requiring distance computations, and, in turn, gaining from this process. We quantify the benefits of this approach implementing it in Apache Spark, the state-of-the-art big data platform.

1.3 Reading Map and Terminology

This introductory chapter provides an overview of the rationale of the work, from the general scenarios and questions that the research period unhearted, to the specific challenges. It also positions the work in the context of previous research.

I delineate in Chapter 2 the general structure of the proposed solution. This chapter is crucial to understand the practical choices made to solve the research questions, the matter of this work. Chapter 2 includes a description of the architecture of LENTA, i.e., the system that comprises IDBSCAN and the evolutionary methodology, called System Knowledge. In this Chapter I also introduce the datasets used in the different part of this work.

Chapter 3 explains the reasons behind the adoption of a particular class of clustering techniques, i.e., density-based approaches. The clustering phase is a building block of the final methodology. This chapter offers a detailed description of the features for the proposed technique, Iterative DBSCAN, showing Iterative DBSCAN capability of outperforming other state-of-the-art techniques also in real-case scenarios. It also presents a solution to the problem of scalability in clustering.

Understanding how the clusters evolve in time is essential for many applications linked to network analysis. It allows observing changes, anomalies, and patterns in data clusters. Chapter 4 presents the methodology used in this work. It clarifies the properties and the implementation of the evolutionary system.

I outline in Chapter 5 the results of the overall system over different real-world settings, showing the capability of addressing different problems, providing a detailed view of the outcomes.

Chapter 6 concludes this thesis, offering final remarks and cues for future improvements and evolutions of this work.

The appendices give an insight into current and future research. Appendix A reports the steps towards a methodology for providing an understandable explanation of clustering results, using Explainable AI (XAI) solutions. Appendix B reports my research contributions, as well as awards, collaborations, and other related activities.

Table 1.1 reports the key nomenclature and acronyms used throughout the thesis. The aim is to provide a common terminology for the most relevant terms appearing in the thesis.

Table 1.1: Summary of the key nomenclature and acronyms used throughout the thesis.

Artificial Intelligence (AI)	According to Stuart Russel and Peter Norvig [67], it can be interpreted as *"the study of agents that receive percepts from the environment and perform actions."*
Class label	Referred to also as label, is a discrete attribute that identifies objects characterized by certain attributes, known as features.
CLUE	Clustering for URL Exploration. It refers to the first work published on this topic [57]
Clustering	According to [2], it is the set of techniques that *" Given a set of data points, partition them into a set of groups which are as similar as possible"*
Command & Control	Infrastructures, usually servers, that manage networks for automatic malicious activities
DBSCAN	Density-Based Spatial Clustering of Applications with Noise. See Sec. 3.2
DGA	Domain Generation Algorithm. A technique used to generate random domain names, dodging static controls based on hostname blacklists.
DNS	Domain Name System. Is a distributed, hierarchical dataset and an application-layer protocol. It is mainly used to translate hostnames to host addresses.
HDFS	Hadoop distributed file system.
Hostname	Is a name assigned to a host. The ones considered in this thesis are fully qualified domain names, i.e., they report the complete domain hierarchy of the DNS.
HTTP	HyperText Transfer Protocol (HTTP), is the Web's application-layer client-server protocol. It defines the messages and their exchange between clients and servers.
HTTPS	Secured HTTP communication. It is characterized by a bilateral encryption.
IDS	Intrusion Detection System (IDS), it is a network or system monitor appliance for preventing malicious activities or violations.
Internet	The global infrastructure that provides services to applications. [39]
ISP	An Internet Service Provider is an organization that provides infrastructures and services for accessing and using the internet.
LENTA	Longitudinal Exploration for Network Traffic Analysis. Acronym for the work presented in [55].
Machine Learning (ML)	Is, according to Tom M. Mitchell's definition [52], the process where *"a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*
Malware	Malicious Software. Applications that aim at harming victims of the attacks that they pursue.
MITM	Man In The Middle. In networking it consists in a relay, and if malicious also in an alteration, of a communication.
NTMA	Network Traffic Monitoring and Analysis. Ensemble of applications related to network management.
Object path	In this thesis it is intended as the path, and the name of a web resource. It is the rightmost component of the URL.
URL	Uniform Resource Locator. It consists of two parts, the hostname, and the object path. It references a resource in the web. It is preceded by the protocol that has to be used to communicate.
Web	World Wide Web (WWW) is the ensemble of resources, identified by URLs, that are accessible throughout the Internet infrastructure.
Tstat	It is a deep packet inspection tool for network monitoring

that exposes information from both TCP and HTTP connections. See section 2.2.2.

1.4 Notes for The Reader

The work presented in this thesis is the result of my research. Therefore, I use the personal pronoun across the manuscript. I report external technologies and methodologies, referencing the sources when I mention and use them. As in most of the research works, the obtained results are an outcome of mutual collaboration, help, and exchange of knowledge. Nevertheless, I want to mention Martino Trevisan, who actively helped in thinking and, most of all, implementing the scalable methodology for clustering proposed in Section 3.5.

Chapter 2

Overview

This chapter intends to provide an insight into the flow of the work of the thesis. The objective is dual. First of all, it aims at detailing the approach followed to solve the previously discussed research questions. Secondly, it points up the different fields of application taken into account for this thesis.

Therefore, Section 2.1 presents the approach. It begins describing the motivations for URL clustering, showing with examples what is the desired outcome. Afterward, it presents a description of the different parts of the system, showing how the different parts and proposed methodologies interact and work together.

Section 2.2 introduces the datasets used through different experiments. It explains the data gathering for all the various collections.

The content of this chapter is part of previously published works. The overview of the system is reported in [55, 56, 54]. The different datasets are present, in various ways and combinations, in all the previous works [57, 55, 56, 54].

2.1 Approach

The approach followed aims to obtain an internet traffic analysis, focused on the study of URLs, in order to group them and extract meaningful information.

In particular, this thesis takes into account both HTTP and HTTPS traffic. An analysis of the latter case is significant since more and more services are or have switched to this communication type. This trend is also actual for malicious traffic. Indeed, while the majority of it still runs on top of HTTP [5], services are moving to HTTPS. [1, 2]

[1] https://www.paloaltonetworks.com/documentation/71/pan-os/pan-os/decryption/ssl-forward-proxy

[2] https://www.juniper.net/documentation/en_US/junos-space15.2/topics/concept/junos-space-ssl-forward-proxy-overview.html

A detailed analysis of results on these two use cases will be addressed in Chapter 5, tackling different perspectives.

This section describes the practical approach followed. Firstly, it details the motivations that guide the selection of specific methodologies. Subsequently, it explains how those methodologies interact together, to solve the research questions of this thesis.

2.1.1 Motivation

The goal of this thesis is to group URLs based on their similarity. The chosen way to achieve this purpose is to leverage string distance to generate homogeneous groups of URLs instead of just merging those elements that have, e.g., a common domain name. In principle, the aim is to strive for grouping together all those URLs that refer to the same service while separating URLs of different services, present some real cases to give the reader the intuition (and the complexity) of doing this.

Table 2.1 shows examples of URLs. *A1*, *A2*, and *A3* belong to the same malware called TidServ – that a professional IDS provided by a leading cybersecurity company spotted in the *HTTP-labeled Dataset* dataset, presented later in Section 2.2.2. All URLs share substrings in the object path, but with strictly different domain names and URLs. This is a common behavior in malicious applications which apply approaches to change the domain name rapidly, with the goal of evading static blacklist-based controls, the so-called DGA (Domain Generation Algorithm) technique, successfully used by several malware programs like *Conficker* [64] and *Torpig* [68], and addressed in various research works [7], [11]. *B1* and *B2* illustrate two URLs generated by Sony connected Smart-TVs which access the same service, but with different URLs. This characteristic is representative of services that employ the same web platform, and that can be interesting to point out. In both the above examples, it is preferable to obtain two groups in output from the algorithm, one for the malware, one for Smart-TV traffic.

It is worth to remark that grouping by domain name is not enough. Indeed, certain domains host logically very different services. This is the case of the third example, *C1* and *C2*, where *Google Flights* and *Gmail* URLs are shown. In this case, the algorithm should identify two groups, one for each service. This work aims at reaching this goal using clustering approaches.

2.1.2 System Overview

Figure 2.1 sketches the overall process. The system processes URLs in batches, each one marked as $UG(i)$, where I insert all unique URLs seen during the *i-th* time interval of duration ΔT. This analysis solely considers unique URLs, extracting them from the logs, since the goal is to understand which resources are fetched by clients, independently of their popularity. At the end of a period, collected URLs are clustered in $\mathscr{C}(i)$. The original log is kept in storage so that it is possible to trace back and inspect other information, starting from the URL.

Table 2.1: Examples of similar URLs

swltcho81.com/[...]VyPTQuMCZiaWQ9[...]	A1
rammyjuke.com/[...]VyPTQuMCZiaWQ9[...]	A2
iau71nag001.com/[...]VyPTQuMiZiaWQ9[...]	A3
bravia.dl.playstation.net/bravia/WidgetBundles/[...]/info.xml	B1
applicast.ga.sony.net/WidgetBundles/SNY_RSSReader/icon.png	B2
google.com/flights/#search;f=TRN,ITT,TPY;t=LAX;d=2018-01-22 ;r=2018-01-26	C1
google.com/mail/u/0/#inbox/160c745d9e5f6684	C2

Several challenges arise here, from the computation of the similarity between two URLs to the proper choice of the clustering algorithm, from the parameter settings to a scalable design.

Once the algorithm outputs the clusters, a sampling process takes care of reducing the dimensionality, i.e., by extracting a summary of URLs found in each cluster, obtaining the sampled cluster $\hat{\mathscr{C}}(i)$. This operation has the benefit to reduce the footprint of the data and to limit the computational complexity of the next steps.

At last, the system compares clusters found in the current batch with those found in the past, which it collects in a structure named System Knowledge and for which the notation used is $\hat{\mathscr{X}}(i-1)$. If there is no match, then the current cluster is considered new and added to the System Knowledge after, eventually, the analyst's inspection, to provide a meaningful label. As the results will show, the availability of several URLs of the same type substantially simplifies the labeling process.

2.2 Datasets

During the evolution of this work, the use of different datasets serves several goals. Furthermore, it allows a broader perspective on the applicability of the overall system. The main focus of this thesis is on network traffic analysis. Specifically, the subject matter is web URLs.

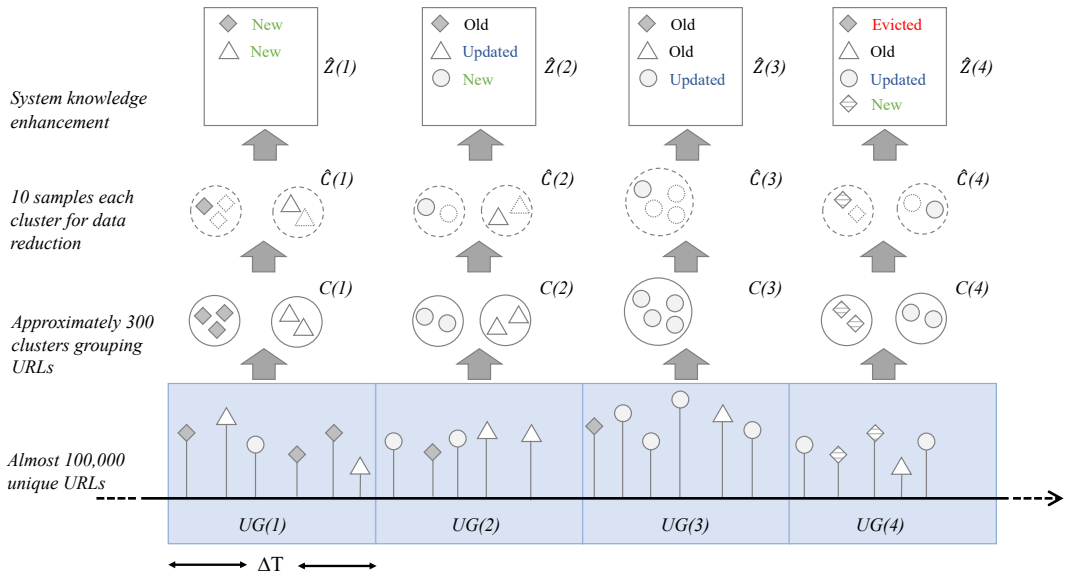


Figure 2.1: LENTA overview. From the bottom, URLs are grouped in batches to then extract clusters. The clusters are sampled and used to update the System Knowledge.

Table 2.2: Overview of the datasets.

Dataset	**Type**	**Elements- Total**	Labels	Days of collection	Users
Synthetic-labeled Dataset	Synthetic	20 000	Yes	-	-
HTTP-labeled Dataset	HTTP	78 421	Yes	1	34
HTTP-not-labeled Dataset	HTTP	1 716 917	No	21	30
HTTPS-not-labeled Dataset	HTTP + HTTPS	800 766	No	14	4

The choice of the datasets aims at covering the related essential use cases and Table 2.2 outlines them. The first collection, *Synthetic-labeled Dataset*, embodies synthetic points in euclidean space. This set functions as test data for algorithm analytics. The second set, *HTTP-labeled Dataset*, is a day of traffic, collected in 2013. It includes knowledge of malicious traffic. The third collection, *HTTP-not-labeled Dataset*, is a three-week generic capture of traffic from households of an Italian ISP. It signifies a representative sample of a household's daily navigation, allowing various observations, including time-related monitoring. The last dataset, *HTTPS-not-labeled Dataset*, consists of a representation of single users web activities. It contains both HTTP and HTTPS traffic through the use of a web proxy installed on every single device.

2.2.1 Synthetic Dataset

For benchmarking and comparison with state of the art algorithms, I generate data using the Python machine learning library *Scikit-Learn*.[3] I rely on the *make blobs* module. It creates globular clusters by specifying the total number of points, the number of desired clusters, the feature space dimension, and the standard deviation of the points in each cluster. The benchmarks require distinct sets. The two synthetic datasets are the following:

(i) variation on the number of clusters, from 10 to 100, while keeping the dataset size equal to 20 000;

(ii) variation on the size of the dataset, keeping a fixed number of clusters to 100 – maintaining the ratio between the number of objects and number of clusters equal to 200.

In both cases, the feature space is three-dimensional, and the standard deviation of points is equal to 0.4.

2.2.2 HTTP - Labeled

Here, I provide an overview of the technologies used to record network traffic and of the tools used to extract useful information. The scenario consists of a sniffer, which passively monitors the traffic generated by a group of hosts, e.g., hosts in a LAN network, or households connected to a Point-of-Presence (PoP) of an Internet Service Provider (ISP). The sniffer is capable of identifying HTTP requests and log them to a file for later post-processing.

In this case, the traffic capture takes place at the PoP of an Italian ISP, where approximately 20 000 customers are connected. Most of them are residential customers accessing the Internet via ADSL modems. The passive probe in the PoP monitors the traffic generated by residential users. The probe runs the passive monitoring tool Tstat [72]. Tstat has been internally developed at Politecnico di Torino, and it is freely available at `http://tstat.polito.it/`. Tstat rebuilds each TCP flow tracks it, and, at the end of the connection, logs a *record* reporting statistics in a simple textual format. When the application protocol is HTTP, Tstat extracts the URL and logs it. In the case of multiple HTTP transactions, due to the usage of an HTTP-persistent option, Tstat logs multiple records. Tstat collects URLs for an entire day, generating more than 100GB of data. For user privacy protection, there is no maintenance of the parameters in the URL, and only saved unique URLs.[4]

[3] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

[4] The usage of this data set has been discussed and approved by my institution ethics committee, and by the ISP security group.

Table 2.3: *HTTP-labeled Dataset* characteristics.

	All hosts	**TidServ infected hosts**
HTTP Flows	267 393	171 863
HTTP Volume	89.99 GB	44.16 GB
Total URL	411 727	255 304
Unique URL	78 421	43 479
Unique Tidserv URL	228	228

The access to a commercial Intrusion Detection System (IDS) allows the labeling of URLs as possibly malicious. The IDS has, at its disposal, an internal database of rules modeling network threats. If some URL matches one (or more) of these rules, the IDS raises an alert and flags the URL with a *Threat-ID*, i.e., a numeric code identifying a specific threat. For this purpose, the relevant signatures are one of a specific malware called *TidServ* (see Section 3.6.1 for a description of the malware). TidServ is known to use polymorphic strings in the URLs to evade detection techniques. The system identified 14 hosts to be infected by the malware in the available dataset.

In the following, I consider one dataset. Table 2.3 provides some statistics about characteristics in terms of volume, number of URLs, and more. The dataset considers traffic generated by the 14 hosts infected by the TidServ malware, i.e., for which the IDS flagged at least one flow as malicious, and 20 additional hosts randomly selected from the population of users; the IDS flags none of the URLs of this second group of hosts. In total, more than 411 000 URLs are present, 78 421 of which are unique. For the sake of completeness, Table 2.3 details also the statistics considering only the 14 infected hosts. Out of the 255 000 total URLs, 43 479 are unique, of which only 228 have the TidServ flag.

2.2.3 HTTP - Not Labeled

The five years of collaboration with the Italian ISP allowed the extraction of large collection of passive traces, using Tstat. For my work, I consider a three-week-long HTTP trace. It is the results of traffic collection in March 2016 in the ISP network, where traffic from more than 20 000 customers was visible. In that period, still more than 40% of traffic was carried by HTTP [37], [71]. I randomly chose 30 users, among the most active ones, i.e., the ones that produce more traffic, collecting almost 2 million URLs for the three weeks. Looking more in detail at the dataset, Figure 2.2 shows that the 30 selected users access every hour several tens of thousands of unique URLs (solid curve - left y-axis) via HTTP, with the total number of unique URLs (dotted curve - right y-axis) that grows to more than 430 000 URLs after one week. The considered users access more

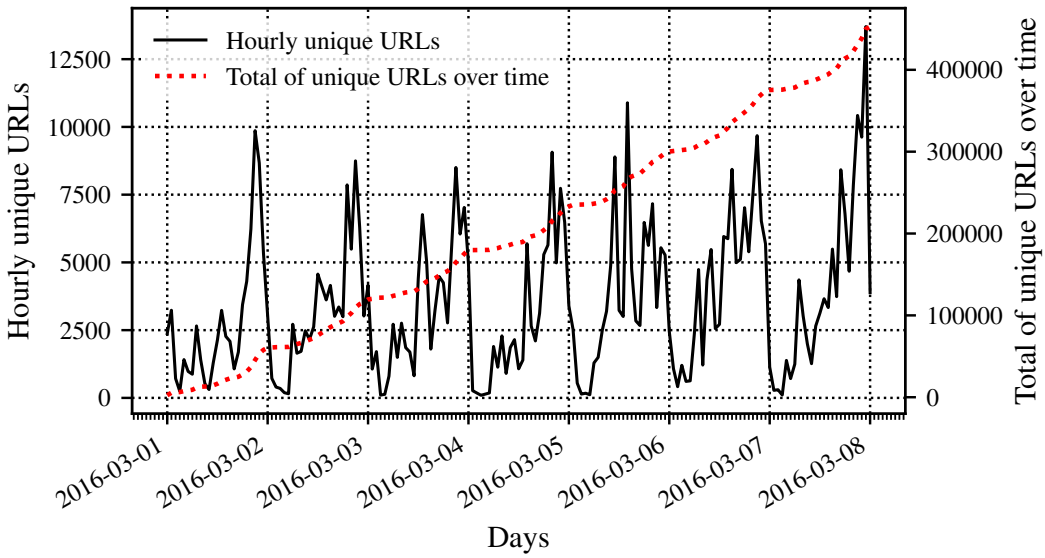


Figure 2.2: Evolution of unique URLs observed on the ISP network.

than 60 000 per day. These numbers give the intuition of the variety of traffic today.

2.2.4 HTTPS - Not Labeled

In this case, ERMES Proxy, [5] a Man In The Middle (MITM) proxy, collects and processes all traffic coming from single hosts. ERMES Proxy is a software module that runs on end hosts. By installing a trusted key, and configuring a system-wide proxy, it gains visibility on all HTTP and HTTPS requests. ERMES Proxy logs all entries and uploads them to a centralized repository on our campus. Volunteers installed the ERMES Proxy for at least one month. The participants' recruitment took place on social networks, specialized forums, and among students at our University. The offer of monetary incentives and the involvement in an experiment aiming at showing them the pervasiveness and danger of web tracking served as motivation. The volunteers have explicitly approved the data collection program, and the project was also subject to a privacy impact assessment, drew together with the data protection officer of our institution. The collection considers two weeks of traffic data, from the most active and dedicated users.

[5] https://www.ermes.polito.it

Chapter 3

Clustering

This chapter intends to answer research questions (i), (ii), (iii), and (iv). It aims at showing the significance of clustering, explaining the choices of the presented implementation, showing the results, and producing a comparison with other well-known clustering techniques.

First of all, Section 3.1 introduces a common terminology for what is intended to be clustering. It describes the principal categories of clustering, helping in then discuss and demonstrate the performed choices.

Subsequently, Section 3.2 details the category *density-based*, which is the one considered in this thesis. It specifies which are the distinctive characteristics that are important to solve the related research questions. Section 3.2 contextually presents the main, prominent algorithms in this category.

Section 3.3 introduces the metrics used to express the pairwise distance between data, both synthetic and URLs. In particular, it details edit distance metrics, used to denote the dissimilarity between strings. This Section discusses and compares different implementations of this class of metrics, analyzing the capability of better expressing the dissimilarity between URLs.

Section 3.4 moves on to the description of Iterative DBSCAN. It reports how this proposed technique differs from the other methodologies considered in Section 3.2 and how it performs the clustering operations. This section includes a comparison of IDBSCAN with the other techniques, using main quality metrics on two different datasets (*Synthetic-labeled Dataset* and one day of traffic extracted from *HTTP-not-labeled Dataset*). Section 3.5 offers a detailed view of the issue of scalability in clustering, including a proposal to solve it.

Finally, in the Sections 3.6 and 3.7, I show in detail what IDBSCAN can extract and provide in output, testing it on two different cases of collections of a day of HTTP URLs, *HTTP-labeled Dataset*, and *HTTP-not-labeled Dataset*.

The work on IDBSCAN is the combination of the outcome of [55, 54, 56]. The issue of scalability in clustering collects the results presented in [23].

3.1 Introduction to Clustering

Clustering or cluster analysis refers to the process of partitioning a set of data into different subsets [69]. Each subset is a cluster.

Clustering is useful as it can guide the discovery of previously unknown groups of objects, all within the collection. Clustering is called data segmentation in some applications, because of the cluster analysis process partitions a large set of data, creating groups based on their similarity. A way of mentioning cluster analysis is also unsupervised learning. Compared to what happens for the classification, in this case, the class label is not present. For this reason, the grouping can be considered a form of learning through observation rather than learning through examples. It also refers to meaningful analysis, obviously, that provides meaning, for the kind of technologies that, starting from the data, are automatically supplied to find classes. Classes are, in fact, conceptually expressive groups of objects that share common characteristics. Cluster algorithms also have several requirements to be met: these factors require scalability and the ability to deal with different types of components, noisy data, incremental updates, arbitrary-shaped clusters, and the ability to meet constraints.

The clustering methods classification proposed by [69] considers the following main categories:

(i) **Partitional methods**: Given n objects, a partition type method constructs k data partitions, where each partition represents a cluster, with $k \leq n$. The partitioning methods perform an exclusive cluster separation. Most partitional methods rely on the concept of distance. Some examples of this type of clustering are K-means and K-medoids;

(ii) **Hierarchical methods**: they create a hierarchical decomposition of a given set of objects. There are two main approaches, bottom-up or top-down;

(iii) **Density-based methods**: most partitional clustering methods are based only on the distance between objects, which means that they can only find spherical-shaped clusters while they encounter difficulty if the clusters are arbitrary. This class of algorithms relies on the concept of density; the idea is to continue expanding a given cluster until the density (the number of objects or points) in the neighborhood of an object equals or exceeds a certain threshold. This method can be used to exclude noisy or outlier objects and discover clusters of arbitrary shapes. Typically, density-based methods consider only mutually exclusive and non-fuzzy clusters. The most prominent method of this class is DBSCAN;

(iv) **Grid type methods**: in this class of methodologies, the object spaces consist of a finite number of cells that form a grid structure. Their main advantage is to guarantee a short calculation time. The time complexity of these algorithms depends just on the number of computed cells and not on the dataset size. Examples of grid-based methodologies are STING and CLIQUE.

3.2 Density-based Clustering

In developing this work, for the problem in question, I considered using this methodological class. As previously introduced, hierarchical and partitional methods are designed to find spherical clusters. Wanting to trace groupings of another type, such as "S-" or oval-shaped ones, the aforementioned techniques would be ineffective and inaccurate; they would identify convex regions where noise and outliers would be included in clusters. This is the reason why a solution to the problem is to model clusters as dense regions in the data space separated by sparse regions. This class of methodologies has other fundamental characteristics useful for this research topic. They include the presence of noise, preventing non-coherent elements from being added to the cluster. They allow the user not to define the number of clusters a priori. Not having to define centroids helps in the replicability of the experiments, the results will always be consistent. Finally, datasets made with elements that can not be traced in Euclidean space can be considered.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) The basic algorithm for density-based clustering is DBSCAN. DBSCAN identifies a cluster as the concatenation of successive dense areas in the data space. Given an object o, it is possible to measure its density by considering the number of elements close to it. DBSCAN finds the core points, i.e., those objects that have dense neighborhoods; then it connects these core points and their neighbors to form the dense regions, i.e., the clusters. The ϵ parameter defines the neighborhood area. This parameter represents the radius of the sphere that has o as the center. A neighborhood is dense if there are at least MinPoints in the sphere of radius ϵ.

Other density-based algorithms have been proposed during the years.

OPTICS (Ordering Points To Identify the Clustering Structure) [6] Ankerst et al. addressed the difficulty of setting DBSCAN parameters and, in particular, the choice of ϵ. OPTICS stems from the basic idea that, given a fixed value for *MinPoints*, the clusters at higher density are contained in the ones that have lower densities. So, the contribution is to compute core points for high-density areas first, finding in that way the denser clusters. OPTICS order points according to their density and provides that list in written or graphical form. Clusters can be identified graphically or automatically from that structure, and so different local densities, i.e., ϵ, may be defined in order to extract clusters in different areas of the data space.

HDBSCAN (Hierarchical DBSCAN) [14, 50] A limitation of DBSCAN is that it is not able to identify clusters made of points that lay at different densities. HDBSCAN aims at solving this problem. It first creates a tree representation of all the possible clusters for different ϵ. The algorithm then solves the problem of finding the best clusters

as an optimization problem, where the overall stability of the clusters, defined following Hartigan's definition of density-contour clusters [32], is maximized. Thanks to this approach, there is no need to tune the ϵ parameter.

CANF (Clustering and Anomaly detection method using Nearest and Farthest neighbor) [22] This method finds the nearest and furthest neighbors to define subgroups of data. In creating the subgroups, it does not need to consider global parameters like ϵ. It computes the radius of subgroups based on the variance of data points, and the number of points in each subgroup and its volume are adaptive to data distribution. Thus, it can identify clusters with different shapes and densities. Since CANF uses subsampling, the time complexity is reduced compared to the methods that use the aggregate data set. However, being iterative, worst-case complexity entails the comparison of all distance pairs again.

3.3 Definition of Distances

The concept of *distance* refers to a specific class of dissimilarity measures that aim at quantifying, with a numeric value, the degree to which two points are far away [28]. The information extracted by the distance computation serves for the clustering step.

Distance is a specific class of *dissimilarity*, which can be defined informally as a numerical measure of the degree to which two objects are different. In particular, a measure of dissimilarity can be called *distance* if it respects three particular properties, which characterize a measure such as a *metric*.

1. **Positivity**
   (a) $d(x, y) \geq 0$ for every x and y,
   (b) $d(x, y) = 0$ just if $x = y$.
2. **Symmetry**
   $d(x, y) = d(y, x)$ for every x and y.
3. **Triangle Inequality**
   $d(x, z) \leq d(x, y) + d(y, z)$ for every point x, y, and z.

For multi-dimensional points space, the Euclidean Distance is the most well-accepted choice. It defines the dissimilarity between two objects as their actual geometric distance.

The distance is thus described as follows:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{3.1}$$

In the URL use case, I look for a distance metric to compute the dissimilarity of strings. Distance measures suitable for application to textual strings take the name of "string metrics" or "string distance functions". The adoption of such metrics is mainly widespread in the field of text-mining, but they find their application in other classes of problems. For example, in medical analysis, when considering DNA traces or in exploratory cases where it is required to compare groups of elements for which one has no a priori knowledge or understanding. Textual distance metrics, therefore, represent a convenient and viable way to represent in numbers the dissimilarity among strings succinctly.

Here, I focus on a particular class of distance metrics, the edit-distance based functions [15]. As the name suggests, the distance between two given strings s_1 and s_2 represents the minimum number of steps required to convert the string s_1 into s_2. Edit-distance functions have been used to target the analysis of free text where strings are well-formed words from a dictionary, with a defined grammatical syntax and with well-understood constraints.

The most popular technique is the *Levenshtein* distance [41] $d_{LEV}(s_1, s_2)$ that assigns a unitary cost for all editing operations, i.e., insert, remove, or replace one character. It computes an absolute distance between strings that is at most equal to the length of the longer string. This approach makes the Levenshtein distance inconvenient when comparing a short URL against a long one, as URL length possibly spans from few to hundreds of characters.

The Levenshtein distance $d_{LEV}(|s_1|, |s_2|)$ is defined as:

$$d_{LEV}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0. \\ min\begin{cases} d_{LEV}(i-1,j)+1 \\ d_{LEV}(i,j-1)+1 \\ d_{LEV}(i-1,j-1)+I(s_{1i} \neq s_{2j}) \end{cases} & \text{otherwise.} \end{cases}$$

where i and j are respectively the lengths of s_1 and s_2, i.e., $|s_1|$ and $|s_2|$, respectively, $d_{LEV}(i,j)$ is the distance between the first i characters of s_i and the first j characters of s_2, and I is the indicator function, namely equal to 0 when $s_{1i} = s_{2j}$.

The Jaro distance follows a different approach. In this case, the distance function considers the number and the order of shared characters between two strings. Let m be the number of matching characters, and t be half the number of transpositions. The Jaro distance $d_{JRO}(s_1, s_2)$ is defined as:

$$d_{JRO} = \begin{cases} 1 & \text{if } m = 0. \\ 1 - \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) & \text{otherwise.} \end{cases}$$

Given the peculiarity of URLs, whose length may vary widely and which may include random substrings, I propose a custom modification of the Levenshtein distance, d_{URL}. URLs are possibly very long strings with up to thousands of characters. For this I explicitly consider the string length, and normalize the results in a [0,1] range:

$$d_{URL}(s_1, s_2) = \frac{d_{LEV^*}(s_1, s_2)}{(|s_1| + |s_2|)}$$

This leads to a bounded distance metric, and specifically $d_{URL} = 0$ if $s_1 = s_2$, while $d_{URL} = 1$ if the two strings are completely different. We count the total number of insertions and deletions, but I weight replacement by a factor of two to count it as a deletion and insertion operation. We call it d_{LEV^*}.

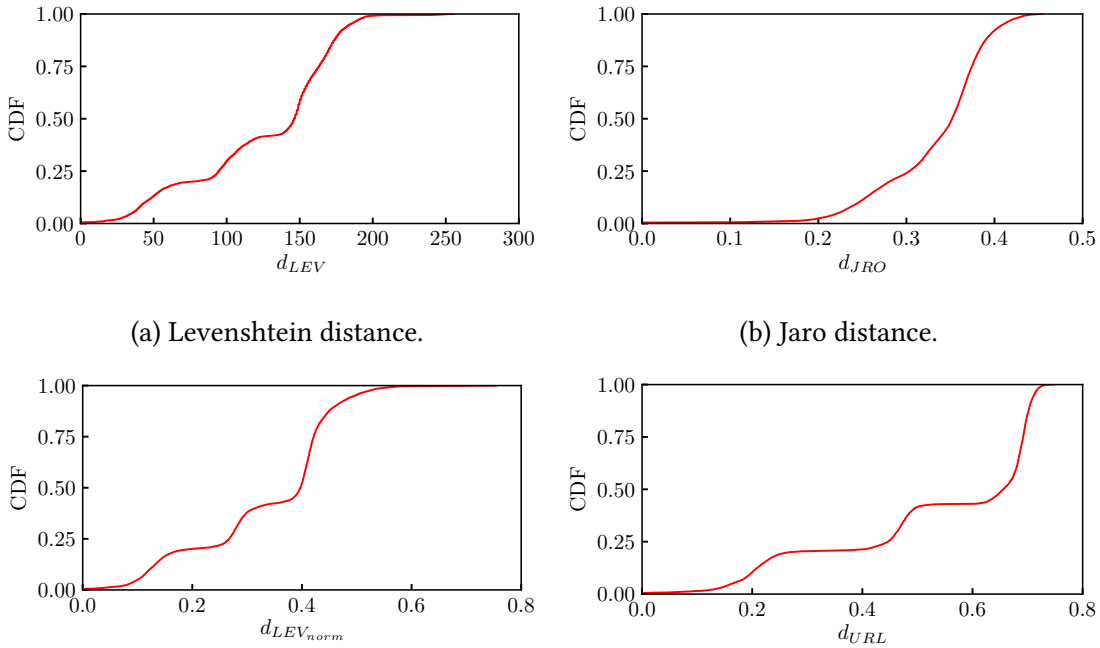
To give the intuition of the different results achievable, consider a simple example. Let s_1 be "google.com" and s_2 be "1goggle.com". We now compute the numerical value provided by each of the considered distance functions. The Levenshtein distance $d_{LEV}(s_1, s_2) = 2$, accounting for one insertion ("1") and one replacement operation ("o" → "g"). For d_{JRO}, the number of matches m is 9 (g,o,g,l,e,.,c,o,m), and the number of transpositions t is 0. Thus $d_{JRO} = 0.094$. $d_{LEV_{norm}}$ is the normalized version for the Levenshtein distance; as I analyzed above, d_{LEV} is 2, thus $d_{LEV_{norm}} = 0.095$. Finally, $d_{URL} = 0.143$ since I have one insertion, weighted 1, and one replacement, weighted 2.

We now run a simple experiment to raise awareness on the importance of choosing an adequate distance function. We consider all the URLs found in my dataset TidServ, a polymorphic DGA malware, has generated that. We then compute the distance between any pair of URLs (u_1, u_2). Fig. 3.1 shows the Cumulative Distribution Function (CDF) of the measured distances for d_{LEV}, d_{JRO}, $d_{LEV_{norm}}$ and d_{URL}, respectively.

Given my goal is to cluster elements that are "close" one to the other, I prefer to have distances concentrated in ranges. A pair of similar elements should exhibit a small distance, while a pair of different elements should exhibit a considerable distance. d_{LEV} shows three groups in its CDF, suggesting potential clusters. However, d_{LEV} support is not bounded in a given range (in my experiments, it spans in the [0:250] range), since it does not entail normalization. This strategy makes the comparison mostly driven by string lengths, i.e., any two short strings will be much more similar than any two long strings. d_{JRO} instead results in a nearly-uniform shape, showing no clear steps that would help in separating close from far away pairs.

$d_{LEV_{norm}}$ and d_{URL} satisfy the intuition of having distance ranges, as clearly shown by three modes in the CDF. Moreover, their support is bounded in the [0:1] range, normalizing the distance for the length of the two considered strings. The results of the two approaches are similar. However, I opt for d_{URL} since, as I can notice comparing Fig. 3.1c with Fig. 3.1d, the latter is able to produce larger values of distances, easing thus the separation of two elements. In fact, $d_{LEV_{norm}}$ range is smaller than d_{URL}. For example, considering the two TidServ elements *T1* and *T2*, where:

The results that I obtain with $d_{LEV_{norm}}$ and d_{URL} are respectively: $d_{LEV_{norm}} = 0.49$, $d_{URL} = 0.70$. The distance obtained with d_{URL} is slightly larger. This feature is however very useful when taking into account URLs that are meant to be different. Considering the *C1* and *C2*, taken from Table 2.1:



(a) Levenshtein distance.

(b) Jaro distance.

(c) Normalized Levenshtein distance.

(d) URL distance.

Figure 3.1: CDFs of distances on the TidServ URLs.

To facilitate the consequent clustering of URLs, I aim at having distances concentrated in ranges.

- **C1:** `google.com/flights/#search;f=TRN,ITT,TPY;t=LAX;d=2018-01-22 ;r=2018-01-26`
- **C2:** `google.com/mail/u/0/#inbox/160c745d9e5f6684`

The results are $d_{LEV_{norm}} = 0.49$, and $d_{URL} = 0.70$. The difference in this case is positively and strongly highlighted, enhancing the contrast between the two elements.

3.4 Iterative DBSCAN

3.4.1 Explaining the Algorithm

In IDBSCAN, the setting of the MinPoints and ϵ parameters is, in general difficult. In particular, MinPoints can be reasonably set using domain knowledge since it represents the minimum number of elements of a cluster. ϵ is instead hard to set, especially if the used distance is not well known. In the first version of my work, CLUE [57], I had to set ϵ by manually tuning it, a cumbersome and error-prone task. Here I propose a

new approach to automatically compute ϵ, while also improving the final clustering. The intuition is to iteratively run DBSCAN, each time using a different value for ϵ, and each time accepting only those clusters that are well-shaped. Objects in badly-shaped clusters are eventually re-clustered in the next iteration, with a different choice of ϵ. This approach produces a remarkable improvement of the clustering stage, by further splitting/merging clusters at each iteration, until they eventually form a well-shaped cluster. After a maximum number of iterations, or in case of a dead loop, the algorithm stops and labels all the remaining elements as noise points (i.e., not assigning them to any cluster). Those are outliers that the system would ignore.

I define ϵ by using an a priori rule, i.e., letting the algorithm cluster a given percentage η of objects at each iteration. To choose the proper ϵ that would guarantee this, I rely on the k-Distance graph rule [2]. Let k = MinPoints. For each object $i = 1, \ldots, N$ in the current batch, the k-th nearest point is found, whose distance is d_i. I next sort $\{d_i\}$ from the lowest to the highest distance, and look for the minimum threshold d_{th} for which $d_i < d_{th}$ for $\eta = 0.75$ (75% of points). I set $\epsilon = d_{th}$. With this choice, 75% of objects have at least k = MinPoints objects at a distance smaller than ϵ. Those would become core points, and form a cluster.

To identify well-shaped clusters, I rely on the *silhouette analysis*, an unsupervised cluster evaluation methodology to find how well each object lies within its cluster [66]. The silhouette coefficient $s(i)$ measures how close the point $i \in C$ is to other points in C, and how far it is from points in other clusters. Let $a(i)$ be the average distance of point i with all points in its cluster. Let $b(i)$ be the minimum among average distance of point i to points in other clusters. In formulas, I have:

$$a(i) = \frac{1}{\|C\|} \sum_{j \in C \neq i} d_{URL}(i,j) \tag{3.2}$$

$$b(i) = \min_{C' \neq C} \left(\frac{1}{\|C'\|} \sum_{j \in C'} d_{URL}(i,j) \right) \tag{3.3}$$

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3.4}$$

It results $s(i) \in [-1,1]$. Values close to 1 indicate that the sample is far away from the other clusters, and very close to all other points in its cluster, i.e., cluster C is very compact. Instead, values close to 0 indicate that i is on or very close to the decision boundary between two clusters. Finally, negative values of $S(C)$ indicate that the clustering process might have assigned i to the wrong cluster. The average $S(C) = E[s(i), i \in C]$ over all points in cluster C is a measure of how tightly grouped all the elements in C are.

Given a cluster C, I say it is well-shaped if $S(C) > S_{min}$, where S_{min} is a threshold for the silhouette values. If C is well-shaped, I insert C in the set of clusters found so far. Otherwise, I put back all points in C in the set of points that I would consider for

Algorithm 1 Iterative DBSCAN

Require: $distance\_matrix$ ▷ The distance matrix is precomputed
Require: $MinPoints$ ▷ Dataset size

1: ▷ Extract ϵ from the k-Distance graph.
2: $\epsilon = select\_automatic\_epsilon(distance\_matrix, MinPoints)$

3: ▷ Compute DBSCAN using $MinPoints$ and the extracted ϵ. Subsequently, extract the silhouette for each cluster.
4: $clusters = DBSCAN(distance\_matrix, \epsilon, MinPoints)$
5: **function** EXTRACT_CLUSTER_SILHOUETTE($cluster$)
6: $S(C) = avg([s(i)\ for\ i\ in\ cluster])$ ▷ i is a point in the considered cluster.
7: **emit** $S(C)$
8: **end function**
9: $clusters\_silhouette = [extract\_cluster\_silhouette(clust)\ for\ clust\ in\ clusters]$

10: ▷ Check the silhouette value for each cluster, if lower than S_{min}, recompute the clustering for the points in that group.
11: **for** $cluster\_silh$ in $clusters\_silhouette$ **do**
12: **if** $cluster\_silh < S_{min}$ **then**
13: $cluster\_distance\_matrix = extract\_cl\_distance\_matrix(distance\_matrix)$ ▷ This function extracts a new distance matrix just with the elements of the considered clusters.
14: $cluster\_\epsilon = select\_automatic\_epsilon(cluster\_distance\_matrix, MinPoints)$
15: $second\_stage\_clusters = DBSCAN(cluster\_distance\_matrix, cluster\_\epsilon, MinPoints)$
16: **end if**
17: **end for**

the next iteration. By setting a maximum number of possible iterations, I avoid dead loops. If the IDBSCAN process is not able to rearrange a cluster, the algorithm labels the contained elements as noise.

Algorithm 1 describes the main steps of IDBSCAN. Firstly, it loads the distance matrix $distance\_matrix$, precomputed in a distributed fashion according to the methodology described in Section 3.5. Using the MinPoints value provided in input, it extracts ϵ, using the k-Distance graph rule at line 2. With the extracted parameters, it computes DBSCAN at line 4.

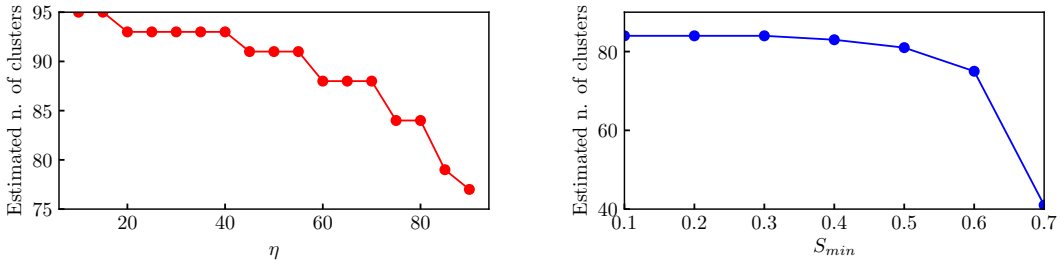
At this point, if each cluster silhouette $S(C)$, computed with the function EXTRACT_CLUSTER_SILHOUETTE at line 5, has a value greater than S_{min}, it is accepted as is. Conversely, Algorithm 1 performed the same procedure followed on the whole dataset, but this time just with the elements of the clusters to reshape. This latter nested stage extracts new clusters that are replacing the old, unfitted ones.

At the end of iterations, I are guaranteed to have all well-shaped clusters, with the

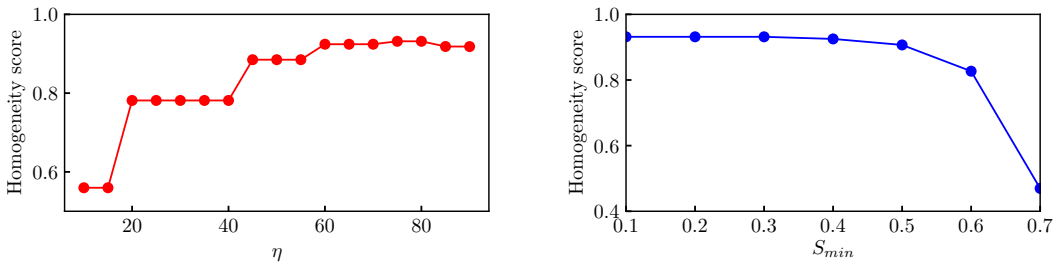
final clustering $\mathscr{C}$ being

$$\mathscr{C} = \bigcup_j \{C_j | S(C_j) > S_{min}\} \tag{3.5}$$

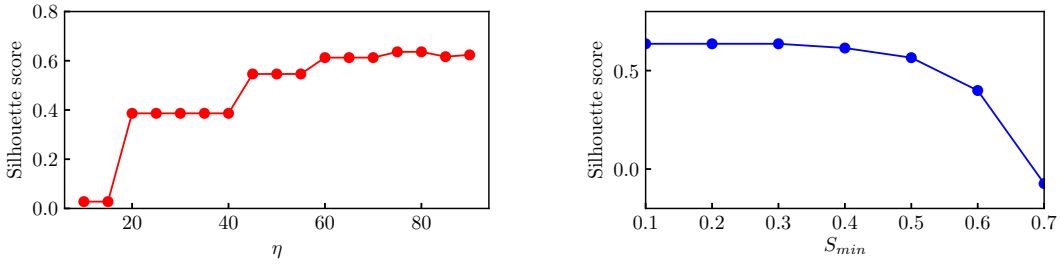
3.4.2 Parameter Tuning



(a) Number of clusters



(b) Homogeneity score



(c) Silhouette score

Figure 3.2: Evaluation of IDBSCAN over artificially generated datasets, varying η (left) and S_{min} (right).

To understand the behavior of IDBSCAN and its sensitivity to parameter settings, I start to test performance over a synthetic set of 20 000 points, which form 100 clusters.

I then vary the η and S_{min} parameters, keeping one fixed, respectively $\eta = 0.75$ and $S_{min} = 0.3$, and varying the other. The Minpoints value is set to 20 in all experiments. For performance indexes, I consider i) the number of clusters, ii) the homogeneity score, and iii) the silhouette. Each configuration has been executed three times, each time with different random points. Plots report the mean value.

Figure 3.2 reports the results. The plots on the left refer to $\eta \in [10,90]\%$, $S_{min} = 0.3$. Results show that values of η higher than 60% generate very pure clusters, with homogeneity score close to 0.9, and silhouette higher than 0.6. The number of clusters is in the 85-90 range (w.r.t. 100 ideal clusters). In a nutshell, IDBSCAN prefers very pure clusters, eventually discarding some few clusters, which turn out to be not very pure.

Right plots report the sensitivity to S_{min} (with $\eta = 75\%$). Any value between 0.1 and 0.5 shows negligible impact, with the performance that suddenly degrades when S_{min} is larger than 0.6. In a nutshell, IDBSCAN is very robust to S_{min}, so that any value in $[0.1, 0.5]$ range would be good.

For the goals of my work, I prefer to give importance to those settings that guarantee homogeneous and pure clusters, well separated to the others. For this, I fix $\eta = 0.75$ and $S_{min} = 0.3$.

3.4.3 IDBSCAN and Other Density-based Algorithms

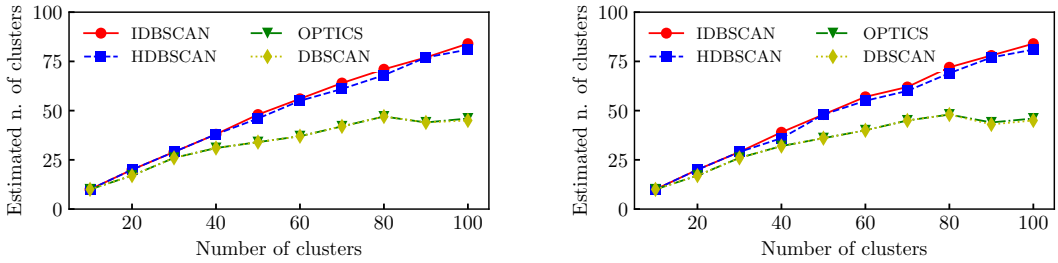
Synthetic Dataset

Here I compare the performance of DBSCAN, OPTICS, HDBSCAN, and IDBSCAN over artificially generated dataset 2.2.1. I use the Scikit-Learn implementations of DBSCAN and HDBSCAN, OPTICS is executed using the pyclustering version, while IDBSCAN uses Scikit-Learn DBSCAN as the building block.[1] For all the algorithms, I use MinPoints equal to 20 and, for DBSCAN and OPTICS, an ϵ value of 0.2.

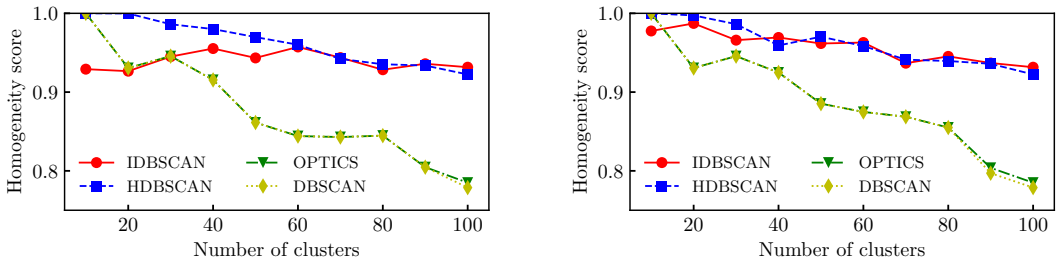
Figure 3.3 reports the results. I focus on two cases: On the left, I keep the total number of points fixed, and I split them over an increasing number of clusters; on the right, I keep the number of points per cluster fixed, and increase the number of clusters.

Results show that IDBSCAN and HDBSCAN outperform both DBSCAN and OPTICS, especially when the number of clusters is high. In those scenarios, HDBSCAN and IDBSCAN can identify a higher number of clusters (top plot), with the number of identified clusters that grows almost linearly with the actual number of clusters. Homogeneity and Silhouette (middle and bottom plots) are very good, too. In this benchmark, where data contains groups of points that are all of the same density, OPTICS performs identically to DBSCAN, given the pyclustering implementation that offers a threshold for ϵ. Not reported here, CPU time of OPTICS is also much higher than the one of HDBSCAN and IDBSCAN, with the original DBSCAN being the fastest, as expected.

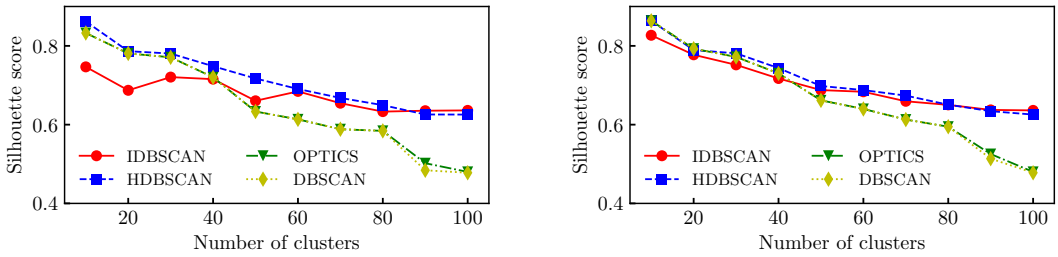
[1] http://scikit-learn.org/, https://pypi.org/project/pyclustering



(a) Number of clusters



(b) Homogeneity score



(c) Silhouette score

Figure 3.3: Evaluation of density-based algorithms over artificially generated datasets. Plots on the left consider a fixed number of points, split over a varying number of clusters. Plots on the right refer to the case where the number of points per clusters is fixed.

IDBSCAN with Traffic Traces

I compute the quality of clustering considering the first day of traffic collected by Tstat, containing 59 543 unique URLs, i.e., considering dataset 2.2.3. I test all algorithms setting the value of $MinPoints = 20$; ϵ, used by DBSCAN and OPTICS, is set at the value $\epsilon = 0.4$, as suggested in [55]. Off the shelf Scikit-Learn implementations of DBSCAN and HDBSCAN are used, OPTICS is executed using the pyclustering version,

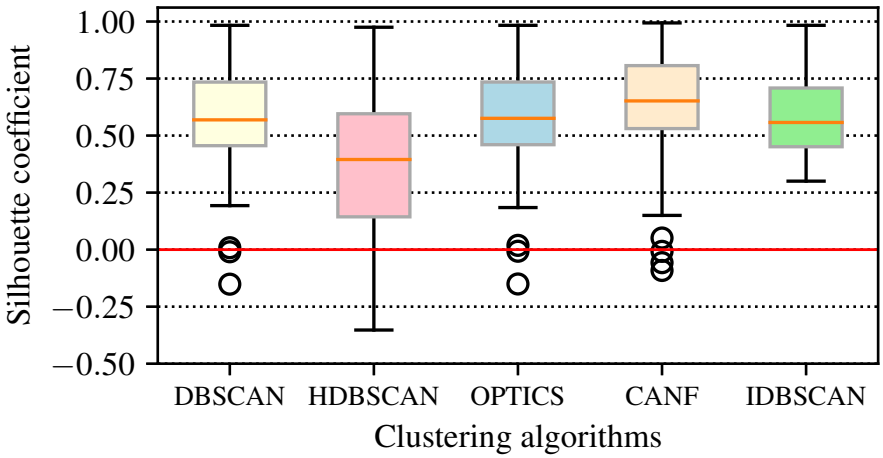


Figure 3.4: Boxplot representing the mean silhouette values for the clusters obtained by each different algorithm on the first day of traffic.

IDBSCAN uses Scikit-Learn DBSCAN as the building block, while CANF uses the authors' developed code.

To evaluate the performance of the algorithms, I report statistics about the clustering results in terms of cluster quality, measured using the silhouette coefficients $S(C)$ obtained for each cluster C in the set $\mathscr{C}$ of extracted clusters. Figure 3.4 depicts the silhouette distribution among all clusters, for each algorithm, using boxplot graph. The orange line depicts the median $S(C)$ value for each algorithm. The colored boxes represent the Interquartile Range (IQR), i.e., the values between Q1 (25^{th} percentile) and Q3 (25^{th} percentile). The black horizontal lines below and above the boxes delimit a larger area, comprised respectively between the *"minimum"*, i.e., $Q1 - 1.5 * IQR$, and the *"maximum"*, i.e., $Q3 + 1.5 * IQR$. Finally, the black circles are the outliers, i.e., all the points that fall out of the previous described areas. In this plot, the y-axis represents the range of silhouette values - recall that silhouette smaller than 0 is an indication of poor clustering - while the x-axis contains the evalued algorithms. Inspecting the boxplots, IDBSCAN shows the best distribution of silhouette between *minimum* and *maximum*, HDBSCAN, the worst. This behavior is given by the fact that IDBSCAN rejects by construction all clusters whose silhouette is smaller than S_{min}. Moreover, the IDBSCAN iterative process splits and re-clusters poorly shaped clusters with variable ϵ. By contrast, the other algorithms produce considerably more variability in silhouette values, since they do not discard poorly shaped clusters.

Table 3.1 provides more details, complementing the silhouette coefficient metric. The results confirm that IDBSCAN appears to be the best algorithm in terms of percentage of URLs clustered, with CANF being the worst. Comparable results are obtainable

Table 3.1: Clustering results obtained applying different density-based algorithms over one day of traffic.

	DBSCAN	HDBSCAN	OPTICS	CANF	**IDBSCAN**
Percentage clustered	45.14	53.16	44.65	29.67	**55.55**
N. clusters	238	563	227	233	**283**
Size largest cluster	15246	4360	15214	15946	**4359**
$S(C)$ largest cluster	-0.15	0.52	-0.15	-0.09	**0.52**
Size smallest cluster	16	20	2	2	**12**
$S(C)$ smallest cluster	0.41	-0.17	0.44	0.84	**0.41**
Mean cluster size	148.28	82.34	205.45	148.28	**147.87**
25% cluster size	5.0	28.0	27.0	5.0	**27.0**
50% cluster size	16.5	41.0	44.0	16.5	**44.0**
75% cluster size	57.75	65.0	89.0	57.75	**83.0**
Computational time (s)	113.70	218.43	8175.82	1500.73	**843.63**

with HDBSCAN, which, however, generates more clusters, thus increasing the analyst work during the inspection phase. The other algorithms generate large clusters of more than 15 000 elements (more than 25% of the data set size), which are difficult to analyze and usually aggregate diverse URL types (as also seen by the low silhouette values). The comparison between clusters' statistics and the silhouette coefficient results in Figure 3.4 helps to shed light on clustering performances. Specifically, the first, second (median), and third quartile of cluster size, they provide clear information on clusters structures. Even if CANF and DBSCAN have a good median silhouette value, they form a large number of small clusters, with the first quartile equal to five. Small clusters are usually denser, influencing thus the silhouette statistics.

Considering execution time, IDBSCAN is slower than DBSCAN and HDBSCAN because of the iterations. Still, the clustering is completed in less than 850 s. In Section 3.5 I provide more details on scalability.

3.5 The Problem of Scalability

Severe scalability issues challenge the use of density-based clustering algorithms for network monitoring. For instance, DBSCAN, the most popular algorithm, has been shown to have polynomial complexity [24]. The main issue of density-based algorithms is the use of expensive ϵ-neighborhood queries that require computing distances among all pairs of records. Some proposals try to overcome such limitations by partitioning the feature space, but this approach is only applicable to points in Euclidean space [16, 46]. They fail when dealing with complex or textual data, such as strings, URLs, or system logs, for which particular distance metrics are required. Edit distance is an example of a metric to compare strings, where Levenshtein distance [41] and other variants are

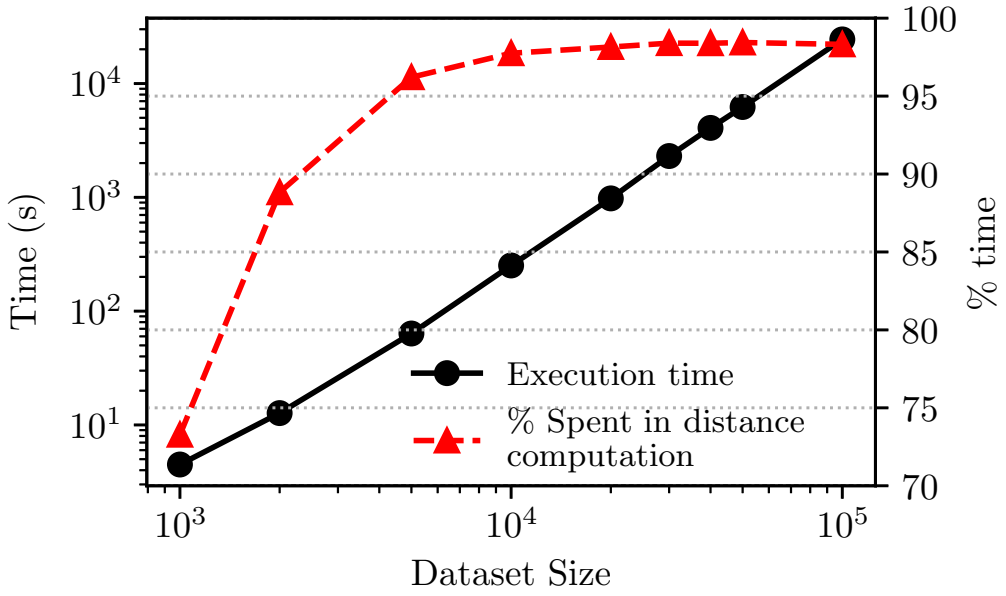


Figure 3.5: DBSCAN execution time and percentage of time spent in calculating distances among strings.

among the most popular. Despite the good results that these metrics provide, they require considerable computation time, which typically scales with the string length, and makes their use in large datasets particularly costly.

To further illustrate the role of distance computation, Figure **??** reports the time spent by the standard implementation of DBSCAN in Scikit-Learn with respect to the growth of the dataset size.[2] Input data are URLs, and the metric used to express their dissimilarity is Levenshtein distance. The execution time (left y-axis in log scale, black curve) exhibits a quadratic growth concerning the dataset size (x-axis also in log scale). As the dataset grows, most of the time is spent on the pairwise string distance computation (right y-axis, red curve).

Considering density-based algorithms, many works propose parallel versions, on both centralized and distributed systems. The authors of [75] propose a parallel version of DBSCAN using the tremendous level of parallelism allowed by GPUs. The authors of [61] design a parallel OPTICS implementation that leverages graph algorithm techniques and builds on the OpenMP high-performance computing platform. Other works propose distributed versions of DBSCAN using Spark and MapReduce platforms [16, 31, 46, 34]. They all partition the feature space and distribute the workload to the executors

[2]Scikit-Learn is a Python library for machine learning: http://scikit-learn.org/stable/documentation.html

to achieve parallelization. However, these works are limited to Euclidean distance metrics, and, thus, cannot handle arbitrary data. It is worth to mention that all these works implement the exact clustering (e.g., DBSCAN or OPTICS) solutions that have been proven to have polynomial complexity with respect to input dataset size. Specifically, DBSCAN complexity is $O(n^{4/3})$ for any dataset with more than two dimensions [24]. Quasilinear time complexity is achieved by authors of NGDBSCAN [44] that propose an approximated version of DBSCAN at the price of lower accuracy. Differently from previous works, I show that calculating distances is the slowest part of density-based clustering when textual data is considered. I claim that, once the distance computation is distributed, it is possible to run centralized versions of the clustering algorithm with no performance penalty with respect to fully parallel algorithms.

3.5.1 Distance Matrix Computation

In my approach, in a first step, I compute the pairwise distances among all couples of elements in the dataset in a distributed fashion, and store them in matrix form; then I run the centralized versions of the clustering algorithms, providing as input the precomputed distance matrix. I first describe the matrix computation and then test and compare the performance of my approach before evaluating the final clustering.

Given a set S of $n = |S|$ strings, my goal is to compute the matrix $D \in \mathbb{R}^{n \times n}$ of all pairwise distances between $s_i, s_j \in S$. I use a modified version of the Levenshtein distance proposed in my previous work [57]. This metric belongs to Edit distance class, which, given two strings s_1 and s_2, measures the variations required to let s_2 be equal to s_1. Differently to the standard Levenshtein distance, results are normalized by the length of the input strings. Note that the time required to compute the Levenshtein distance typically increases with the string length. Given two strings of size l_i and l_j, the Levenshtein edit distance scales with complexity $O(l_i \cdot l_j)$.

To compute distances in a distributed fashion, I build on Apache Spark to distribute the workload on several executor nodes [3]. Some ingenuity is required here. I consider two possible solutions.

The first simple solution splits the matrix S into k rows, and then "maps" the computation of each row among executors. With $k = n$, each executor would compute all distances from one string s_i to any $s_j \in S$. Rows are then collected to build D, which is stored on disk. This solution, however, suffers from the fact that executors that are assigned a long string s_i would become the bottleneck easily since the length of s_i heavily influences the computation of the distance.

The second smarter solution instead generates all possible pairs (s_i, s_j) and "maps" the computation of each pairwise distance. Here, in the first stage, executors generate

[3] The code is public and available at: https://github.com/marty90/spark-distance-matrix

Algorithm 2 Distance Matrix computation on Apache Spark

Require: $S = \{strings\}$ ▷ Input dataset S of strings
Require: n ▷ Dataset size
Ensure: $matrix\_rdd$ ▷ The $n \times n$ distances

1: ▷ Create a RDD containing all possible pairs of elements.
2: $index\_rdd = parallelize(\{0,1,...,n-1\})$
3: $pairs\_rdd = index\_rdd \times index\_rdd$ ▷ Cartesian product

4: ▷ The $COMPUTE\_DIST$ function calculates the distance between pairs of elements given the $strings$ dataset.
5: **function** COMPUTE_DIST(i, j)
6: $d = edit\_distance(strings_i, strings_j)$
7: **emit** $(i, (j, d))$
8: **end function**
9: $distances\_rdd = pairs\_rdd.map(COMPUTE\_DIST)$

10: ▷ Rebuild the distance matrix grouping and sorting pairs for a specific row.
11: **function** REBUILD($tuples$)
12: $sorted\_tuples = tuples$ sorted by first element
13: $sorted\_dist = [d\ for\ i, d\ in\ sorted\_tuples]$
14: **emit** $sorted\_dist$
15: **end function**
16: $matrix\_rdd = distances\_rdd.groupByKey()$
17: $matrix\_rdd.map(REBUILD)$
18: **emit** $matrix\_rdd$.sort()

all the element pairs in parallel, and the resulting list is automatically split across nodes by using the shuffling mechanisms of Spark. In the second stage, executors compute the distances for the pairs. In case an executor gets stuck in the computation of one pair involving very long strings, other executors can still consume other pairs. Finally, the matrix is rebuilt and stored on disk. The resulting algorithm is thus much less sensitive to the way data is split among nodes. Algorithm 2 shows the pseudo-code of the second solution. I leverage the specific Spark feature to compute the Cartesian product automatically to generate all string pairs (line 3). The function $COMPUTE\_DIST(s_i, s_j)$ computes and emits the distance between s_i and s_j (lines 4-8). Results are then first grouped by key, i.e., the string s_i, to for a row of D (line 16). Later rows are re-ordered and aggregated by the function $REBUILD()$ (lines 11-15). The real implementation actually performs an extra optimization computing distances only for the upper triangular matrix and mirroring them to the lower triangle.

For the sake of comparison, I also compute distances in a centralized fashion, and

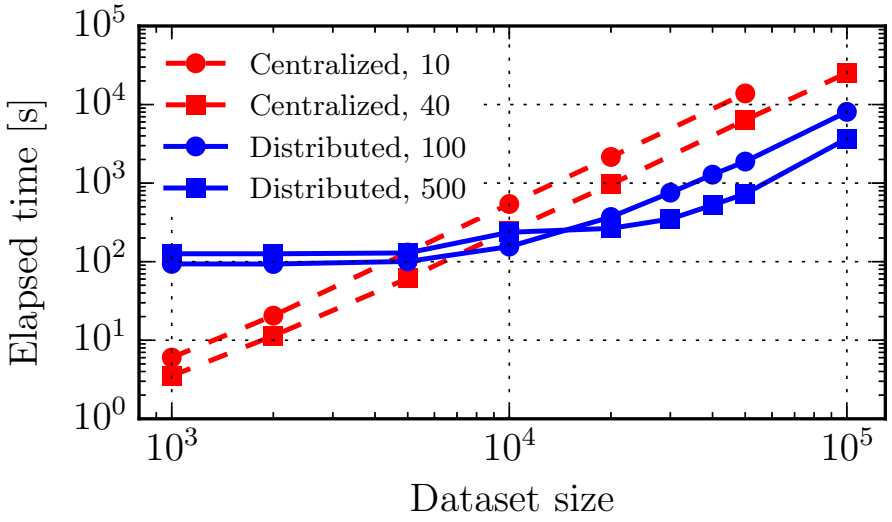


Figure 3.6: Elapsed time in distance matrix computation.

profit from the Scikit-Learn Python library, using the `pairwise_distances` function that allows parallelism by splitting the workload on multiple jobs (see Scikit-Learn glossary for details), i.e., exploiting the vertical scalability offered by multi-core CPU architectures.

3.5.2 Distance computation

This analysis attempts to investigate the cost of calculating the *pairwise distance* for all the pairs of input elements, producing, as a result, the final *distance matrix*. Given n records, $n \times n$ distances have to be computed, allowing clustering algorithms to perform the ϵ-neighborhood queries.

Benchmarking Datasets For the experiments, I use the dataset *HTTP-not-labeled Dataset*, extracting the URLs from the first two days of activity. The dataset includes more than 100,000 unique URLs. To build smaller datasets, I randomly split the original 100,000-URL set. I also perform experiments with a set of different URL lengths to evaluate the impact of the computational time of the Edit distance.

Experimental Platform For my experiments, I rely on two different systems. Centralized experiments are run on a high-end server equipped with two Intel® Xeon® E5-2640 processors providing 40 cores in total and 128 GB of RAM. Experiments with Apache Spark run on a medium-sized Hadoop cluster composed of 25 worker nodes with 564 cores and 2TB of RAM overall. I use Spark version 2.3.0, and all code is written in Python for a fair comparison.

Figure 3.6 depicts the elapsed time – i.e., the amount of time needed for the job to complete – for the distance matrix computation while varying the dataset size, both with a centralized (red dashed lines) and distributed approach (solid blue lines). I consider two different numbers of jobs (10 and 40) and executors (100 and 500). Results

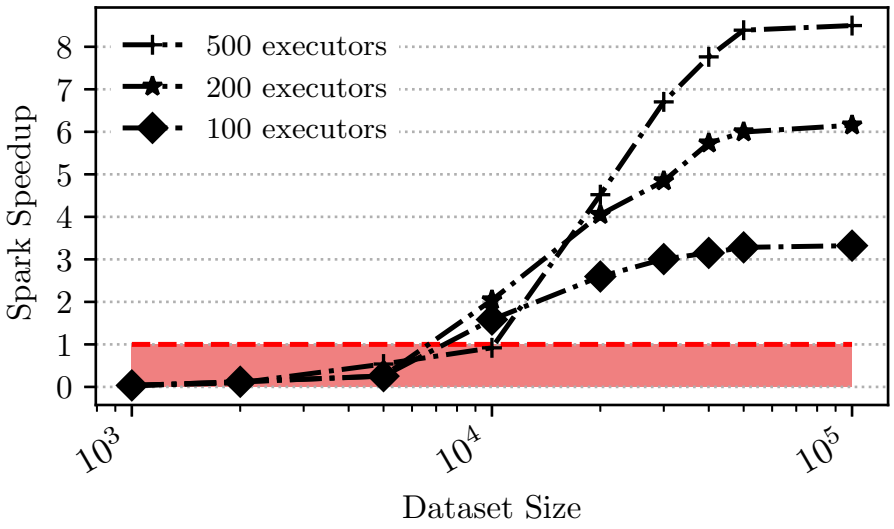


Figure 3.7: Speedup factor of Spark distributed approach varying the number of Spark executors w.r.t. centralized algorithm using 40 threads.

point out the differences between the two methods: the centralized approach shows a clear, direct relation with the dataset size, and grows with a $O(n^2)$ complexity (note the log-log scale). With the broadest dataset (100 k URLs), 40 jobs on a single machine require more than 7 hours to complete the operation, while the 10 jobs configuration does not reach completion in a reasonable time. On the other hand, the Spark-based version takes less than 1 hour, showing the goodness of the horizontal scalability approach for a large and complex dataset. Note indeed that for datasets smaller than 5,000 URLs, the overhead caused by the initialization of the executors and the shuffling of results impact the distributed solution, making it slower than the centralized one.

For both the centralized and distributed approach, the increase in the number of jobs and executors guarantees a better utilization of resources, providing better overall scalability, and proving that the distance matrix computation is amenably parallelizable. Specifically, with Spark, the overall job time remains practically constant up to when the cluster capacity is reached. The cluster capacity depends on the number of used executors, and on the size of the cluster. In particular, with more than 10,000 elements, the complexity becomes again $O(n^2)$, meaning that all the computation power of my system has been saturated.

To better highlight results, Figure 3.7 depicts the speedup factor, computed dividing the execution time of the centralized implementation (40 jobs) by the distributed one (100, 200, and 500 executors). Results allow to appreciate better that: (i) the speedup is less than 1 when dataset size is small (red area); (ii) when the size is big, i.e., the input dataset contains more than 10,000 URLs, the speedup factor increases significantly. Increasing the number of executors has a noticeable impact on the speedup factor growth, which is shown to reach a value up to 8 when Spark exploits 500 executors. With extensive datasets, the communication overhead slows down the speedup. In Figure 3.6 and

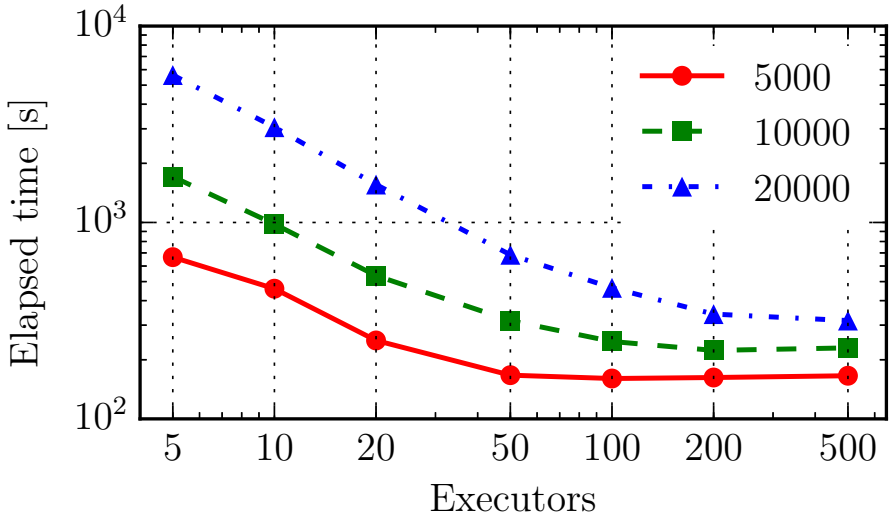


Figure 3.8: Distance matrix computation time with different number of Spark workers.

in Figure 3.7 it can be noticed that the performance obtained using 500 executors on small datasets are worse than what obtainable with lower parallelization, that because of the initialization cost. The results are better for dataset sizes greater than 10,000.

I now concentrate on the impact of different parallelization levels, measuring the time required for the matrix computation when varying the number of executors from 5 to 500. I consider three different dataset sizes, n =5 000, 10 000 and 20 000 URLs. In Figure 3.8 one can appreciate the effects of greater parallelization in the execution time (notice the log-log scales again). However, one can notice the curve flattening for the values of 200 and 500 executors, where the benefits of higher parallelism are nullified by the communication and synchronization overhead.

Lastly, I investigate the impact of the string length on the overall computation time. To quantify this phenomenon, I conduct experiments considering sets of URLs lying in different length ranges. I define 11 bins and divided original URLs according to their length until 10 000 elements composed each bin. I used the 40-jobs configuration for the centralized approach and 500 executors for the distributed one. Figure 3.9 reports the time elapsed for computing the entire distance matrix for each set of different length URLs. The results demonstrate the impact of the string sizes and, again, the different behaviors of centralized and distributed approaches. Note how, in the centralized case, the computation time is highly dependent on the URL length, with a penalty incurred with extremely long strings for which the distance computation becomes the main driver. The distributed approach instead can better exploit the higher parallelism, so that overall time decreases. Again, for particularly short strings, the initialization and communication overhead is still dominating the computation.

The experiments in the computation of the distance matrix show the potential of the horizontal scalability, which allow a dynamical allocation of resources, enabling an enhancement in performance without negatively influencing the users [4]. At the same

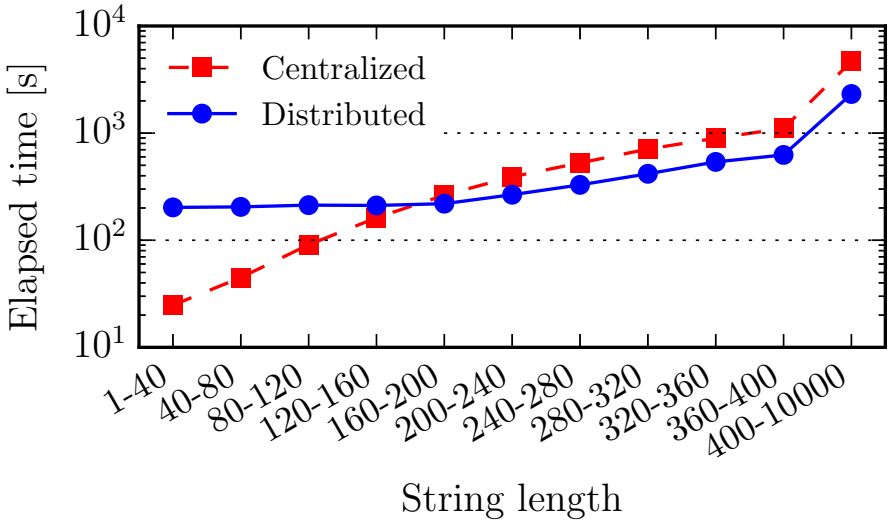


Figure 3.9: Distance matrix computation time for sets of different string length, with $n = 10\,000$ strings in each set.

time, when dealing with small data, a centralized solution. Indeed, with respect to the centralized solution, it requires an initial setup time and communication between the nodes.

3.5.3 Clustering Algorithms Computation

Once the distance matrix D has been computed, it is possible to run the desired clustering algorithms, tune their parameters, and compare the results. To show this, I now focus on the execution time of the clustering process, when the pre-computed distance matrix is provided as input. I consider five possible clustering algorithms that I briefly introduce next, before running experiments. They all require to compute all pairwise distances among points.

All algorithms were tested by setting the value of $MinPoints = 20$; the additional parameter ϵ, used by DBSCAN and OPTICS, is set at the value $\epsilon = 0.4$ and starting from a pre-computed matrix D. My aim here is to show the possible improvement in terms of total execution time and the flexibility in the algorithm choice that the pre-computation of the distance matrix produces. Off the shelf Scikit-Learn implementations of DBSCAN and HDBSCAN are used, OPTICS is executed using the pyclustering version, IDBSCAN uses Scikit-Learn DBSCAN as the building block, while the authors fully develop CANF.

Figure 3.10 shows the execution time of the clustering algorithms, using the off-the-shelf implementation offered in Scikit-Learn. The dark gray area indicates the best execution time for the distance matrix, as obtained from Figure 3.6. The light gray one represents the computation time with the centralized approach. As it is visible, for most of the algorithms, the execution time is an order of magnitude lower than the time needed for the mere computation of the distance matrix (note the log-log scales). Even

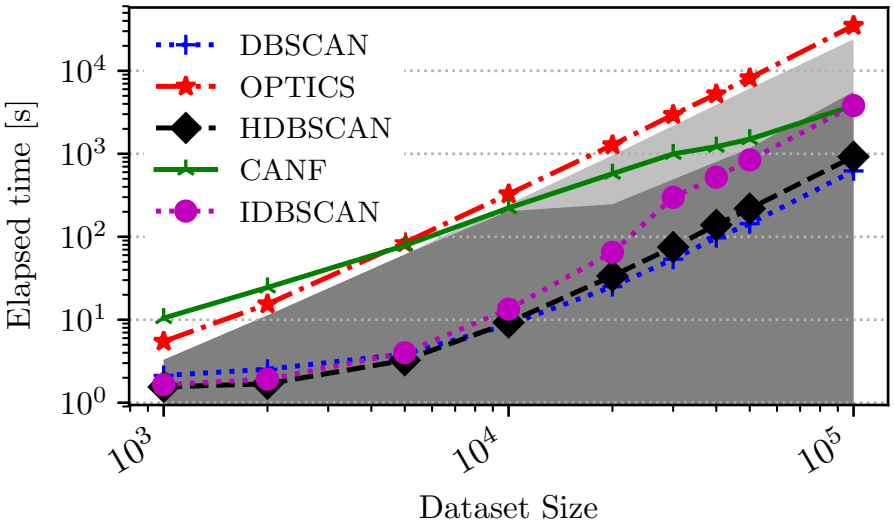


Figure 3.10: Algorithms execution time in seconds.

CANF benefits from the distributed computation of D for large datasets. The clustering stage would bottleneck only OPTICS. DBSCAN and HDBSCAN obtain the best performance. Despite running on a single thread, both end the computation in several orders of magnitude less than the time to compute the matrix $\boldsymbol{D}$, even when extensive datasets are involved. The heterogeneity in the obtained results reflects the variety in the chosen algorithm implementations. Each of them, indeed, takes advantage of a different degree of optimization. Regardless of time performance, in this thesis, I opted for restricting the field of investigation to three widely-known and applied algorithms (i.e., DBSCAN, OPTICS, and HDBSCAN), and to two (i.e., CANF and IDBSCAN) designed explicitly by the authors having the URL clustering problem in mind. Such choice results also in different clustering performance, whose analysis is out of the scope of this work.

3.6 Malware Detection

This section presents experiments conducted on the *HTTP-labeled Dataset* dataset. The results show the capacity of the proposed solution to capture and highlight the anomalous behaviors carried out by malicious activity in a completely unsupervised manner.

3.6.1 TidServ Overview

This section firstly provide some highlights on the traffic produced by *TidServ*, a popular Trojan Horse also known as Alureon, TDSS or TDL [9]. After infecting a host and transforming it in a bot, this malware communicates with a Command-and-Control

(C&C) server to receive commands.[4] Communications with C&C servers are typically established using HTTP, so to evade firewalls. Initially, communication was taking place using static URLs. In this scenario, security software could easily block the communications using, e.g., static rules, and blacklisting. However, the malware started to evade such rules by using polymorphic approaches successfully, e.g., randomly generating and rotating hostnames for C&C servers, or adding randomness in the URL path. This evolution in the approach makes the compilation of static blacklists based on string matching a more difficult task and in turn, less effective.

TidServ adopts this expedient by changing the URLs periodically to contact C&C servers. To give the reader the intuition of how random a TidServ URL can appear, Table 3.2 reports four examples of URLs that the IDS flagged. Hostnames and paths change, but some common parts (in bold) are still visible. In some cases, the shared pattern may be very long, but in others as few as four characters are found in common, suggesting that different communication patterns may be present. Observing these patterns is easy if one is provided the correct set of URLs, but spotting them when mixed in the hundreds of thousands of URLs generated by a host makes the detection very challenging.

3.6.2 IDBSCAN Execution Over Labeled Dataset

I run IDBSCAN over the dataset, including one day of HTTP traffic from 34 hosts, 14 of which are flagged by the IDS as infected by the TidServ malware, from the dataset *HTTP-labeled Dataset* presented in Section 2.2.2. Afterward, I check those clusters which contain at least one TidServ URL. In total, IDBSCAN identifies 7 clusters according to the distribution reported in Table 3.3. Each cluster contains URLs not originally flagged by the IDS. By manually checking those, it confirms that IDBSCAN correctly assigns those to TidServ clusters, being those very likely to be false negatives for the IDS (i.e., the IDS did not flag those despite being malicious). The table shows the polymorphic behavior of the malware. Several hostnames are used for the C&C server, and randomness is introduced in the URL paths too. IDBSCAN is able to identify all the TidServ labeled URLs in the entire set of 78,421 URLs. For instance, consider the first cluster, being the largest with 192 URLs pointing to resources hosted on 14 different hostnames, whose shared pattern is only the `.com` substring. Interestingly, IDBSCAN groups together 118 different URLs that the IDS flags as TidServ, and 74 additional URLs not flagged by the IDS. Table 3.4 details cluster 7. In this case, only the first (in bold) URL out of 37 is flagged as malicious by the IDS. The similarity within all URLs is, however, clear. I therefore conclude that those are false negatives for the IDS. This clear example explicates how IDBSCAN could be used to support the generation and update process

[4]The C&C servers run on central computers that attackers use to update, instrument and control infected hosts.

Table 3.2: Examples of TidServ URLs flagged by the IDS. Common substrings in bold.

swltcho81.com/NZf4A07d7r7yE1C1dm**VyPTQu**MCZiaWQ9YjZjYWVhNj E0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9 NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**XV pbmZlIG15ZGVzaw==38c
rammyjuke.com/kaI1wWRd8Y5yfbU9dm**VyPTQu**MCZiaWQ9YjZjYWVhNj E0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9 NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**WZ vcnVtIGFybWF0YSBkZWxsZSB0ZW5lYnJl37g
bangl24nj14.com/TVq2BttP743qt1c8dm**VyPTQu**MCZiaWQ9YjZjYWVh NjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9 NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**XV pbmZlIG15ZGVzaw==05c
iau71nag001.com/Kvb13nWd6P4XrFs3dm**VyPTQu**MiZiaWQ9MDU0NWQw ZDQwY2MyODU4YWNjYzFlZjJkM2FiZDA5N2RiYmRlYmVkZiZha**WQ9N** TAwMTg**mc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**WZhY2 Vib29r27c

Table 3.3: TidServ clusters identified by IDBSCAN.

ID	Tot. URLs	TidServ URLs	Hostnames	Most common hostname
1	192	118	14	81hja01aala.com
2	79	75	1	wuptywcj.cn
3	32	18	2	clickpixelabn.com
4	6	6	1	biiwf3iidpkxiwzqmj.com
5	6	5	1	zl091kha644.com
6	5	5	1	zhakazth.cn
7	37	1	3	lkckclckl1i1i.com

for IDS signatures.

Table 3.4: URLs in cluster 7. The IDS flagged only the first.

gnu4oke0r.com/4...PTQuMCZiaWQ9NWJjNWFiMjE1Yj...5pdCZxPWxvdWlzIGNydWlzZXM=16h
lkckclckli1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==27g
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==27g
lkckclckli1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==26g
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==26g
lkckclckli1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==18x
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==18x
etc.

3.7 IDBSCAN on One Day of Unlabeled Traffic

In this section, I provide experimental results on a general use case. I choose $\Delta T = 24\,\text{h}$, $\eta = 0.75$, $S_{min} = 0.3$, $p = 0.2$ and $MinPoints = 20$ to look for well-shaped and big enough clusters. I tested different parameters, observing little changes. Experiments are not reported here for the sake of brevity.

I start to analyze the first day of traffic. As seen from Table 3.1 in Section 3.4.3, IDBSCAN obtains 283 clusters from the set of 59 543 original unique URLs. The Silhouette coefficient $S(C)$ has a value of 0.5 or more for 183 clusters, with 55 of them with $S(C) > 0.75$. That is, clusters result very well-shaped.

The top part of Table 3.5 shows the most massive clusters, while the bottom part displays those with the highest silhouette. The table reports the silhouette $S(C)$, the most common hostname in the cluster (in brackets the total number of distinct hostnames), the number of unique URLs, and the type of the service. Although the majority of clusters are relatively small, some contain a considerable number of distinct URLs and different hostnames. That behavior is not to be taken for granted, as often the complexity of URLs structure tends to increment the distance also for actually similar elements.

After this stage is already possible to identify some suspicious clusters, for instance, 30 unique URLs form a cluster where URLs have all the same IP address `219.129.216.161` – but random paths. After further analysis[5], this cluster is indeed found to be malicious. Other suspicious clusters emerge as well. At last, it is essential to mention that the same service, *i.e.*, the same hostname, may be broken apart in multiple clusters, each one containing specific content. For example, from the analysis of other clusters, it results that the Chinese messaging system `msg.71.am` finds its representation into two clusters, one serving images (.GIF), and the other exchanging control information like device reports.

These results clearly show that IDBSCAN let the services that commonly characterize the traffic emerge. The security analyst can then analyze clusters and easily label them.

[5]Google results: https://goo.gl/q3DgT8; VirusTotal results: https://goo.gl/fqrNkG

Table 3.5: Insight of the clustered HTTP traffic from the first day of analysis. On the top, the largest clusters. On the bottom, the top well-shaped clusters.

$S(C)$	**Main hostname (unique hostnames)**	**Elements**	**Activity**
0.52	scontent-mxp1-1.cdninstagram.com (4)	4359	Instagram CDN
0.92	se-rm3-18.se.live3.msf.ticdn.it (6)	3504	Entertainment - Streaming CDN
0.36	skyianywhere2-i.akamaihd.net (9)	2087	Entertainment - Streaming CDN
0.30	www.google-analytics.com (29)	1940	Tracking
0.95	rtinfinityh2-a.akamaihd.net:80 (1)	1227	Entertainment - Streaming CDN
0.76	videoassets.pornototale.com (1)	751	Adult content
0.57	tracking.autoscout24.com (2)	592	Tracking
0.37	ec2.images-amazon.com (10)	575	Image CDN
0.56	thumbs-wbz-cdn.alljapanesepass.com (1)	393	Adult Content
0.66	video-edge-8fd1c8.cdg01.hls.ttvnw.net (4)	359	Entertainment - Streaming
0.98	iframe.ad (1)	27	Advertising
0.97	news.biella.it (1)	23	News
0.95	rtinfinityh2-a.akamaihd.net:80 (1)	1227	Entertainment - Video Streaming CDN
0.93	motoitalia01.wt-eu02.net (1)	45	Tracking
0.92	skygo.sky.it (1)	45	Entertainment - Video Streaming Platform
0.92	se-rm3-18.se.live3.msf.ticdn.it.msf.ticdn.it (6)	3504	Entertainment - Video Streaming CDN
0.92	219.129.216.161 (1)	30	Malware
0.92	a.applovin.com (1)	20	Analytics
0.92	rum-dytrc.gazzetta.it (1)	47	Entertainment - Analytics

3.8 Conclusion

This Chapter presented IDBSCAN, a clustering solution based on the iteration of DBSCAN, to improve the quality of the results. Sections 3.6 and 3.7 showed the applicability of this methodology over two interesting real-world cases. IDBSCAN exhibited the capability of offering an easy way of exploring traffic data, through the formation of well defined clusters where similar URLs are grouped. Thanks to the cohesiveness of the formed clusters, clues about the processed traffic easily emerge and let the analyst identify possibly malicious traffic or advertisement, tracking and third-party services.

While we clearly showed the effectiveness of our approach by presenting produced results on a real but small dataset, an additional effort is still required to make the

system scale at runtime usage, in order to enhance the effectiveness of the analysis and the discovery of anomalies, which would be difficult to spot in a single day exploration. This topic will be discussed in Chapter 4, and the application of the system to real-world use cases will be described in Chapter 5.

Chapter 4

Evolution and System Knowledge

This chapter presents the study for the generation of an evolutionary analysis of the URLs traffic produced during a period, across different users. The evolutionary analysis implies the use of different techniques, tools, and methodologies. The idea behind this system is to create a record of all the clusters visited for each snapshot, then use clustering each day and compare the newly generated clusters with an activity record, with a solution called System Knowledge.

Firstly, Section 4.1 presents System Knowledge. It is a core part of the overall system. This section displays the different parts and introduces the following sections.

Subsequently, Section 4.2 describes the sampling stage, that must be performed immediately after the clustering step, to mitigate the storage consumption and facilitate the comparison between clusters. This section describes the considered sampling methodologies, including the proposed one. It reports sampling tests on both synthetic and real-case datasets, showing the effectiveness of the introduced methodology.

Afterward, Section 4.3 describes how to compare the new clusters, extracted in the last observation of data, with the ones previously obtained, from an analysis of the similarities between the *new* and *old* clusters. It describes the performance of the System Knowledge in a controlled experiment, obtained considering peculiar URLs.

Section 4.4 reports a procedure called ageing. It is a fundamental step to keep the System Knowledge up to date. After the comparison, if a new cluster is similar to some System Knowledge cluster, it is necessary to update and interchange the elements representing the latter, to follow the evolution of the comprised category. Finally, Section 4.5 describes pruning techniques. These techniques are necessary to remove old, not visited clusters, and keep the System Knowledge clean. A technique based on a sliding window determines the *life* of the System Knowledge clusters.

The evolutionary approach described here is the current output of the works presented in [54, 56].

4.1 What is the System Knowledge

This section aims at describing the structure and implementation of the System Knowledge. System Knowledge $\hat{\mathcal{Z}}$ is a data structure, which contains a collection of groups $\hat{Z}_l \in \hat{\mathcal{Z}}$. These groups contain, at least initially, a representative subset of URLs in a cluster. Besides, they have at least two metadata, the date of addition of the group in the data structure, and the date of the last appearance. Indeed, as described in Section 2.1, at each clustering step, the new clusters $\hat{C}(i)$ are compared with the group in the System Knowledge. If the groups are similar, the group in the System Knowledge is updated, also updating the elements contained in the reference group, as explained. Another possible meta-data is the group of ID(s) of the user(s) that generated clusters containing that type of content, even at different times. Depending on the application case, this information may be present or not. HDFS is the storage structure for the System Knowledge since its size can grow in time.

The comparison of each newly found cluster $\hat{C}_j(t)$ with those present in the System Knowledge $\hat{\mathcal{Z}}(t-1)$ is a CPU intensive operation. Also, this involves the computation of edit distances between many URL strings. Therefore, the System Knowledge module runs in the Spark environment to guarantee scalability, both in terms of storage and performance. The idea again is to first compare in parallel the clusters $\hat{C}_j \in \hat{\mathcal{C}}(t)$ with $\hat{Z}_l \in \hat{\mathcal{Z}}(t-1), \forall j, l$. In a second step, perform the insert and update operations on the System Knowledge representatives.

Using a tree organization - through the implementation of the Vantage-point tree (or VP tree) [78] - for the representatives in $\hat{\mathcal{Z}}$, improves look-up of the most similar cluster in the System Knowledge (Equation (4.3)). The VP-Tree structure is replicated in each executor and used to compute the k-NN (with $k = 1$) for each representative of each cluster $C_i(t)$. In the end, the algorithm updates $\hat{\mathcal{Z}}$, and build and redistributes the newly updated VP-tree. This technique speeds up the process and improves its scalability. Considering the applicative scenarios discussed in Chapter 5, computing and maintaining the System Knowledge for the two weeks of HTTPS traffic - apart from the clustering computation - requires 34 minutes, on average circa 2 minutes and 30 seconds per day. Considering the three weeks of HTTP data, the whole process takes less than five hours, circa 14 minutes per day.

4.2 Sampling

Once the clustering algorithm returns the final clusters, it is necessary to sample a subset of elements from each of them. The rationale is twofold: to ease the comparison between clusters by reducing computational complexity while maintaining their information quality; and to keep in the System Knowledge a digest of the collected traffic, thus reducing its footprint.

I sample each cluster $C_j \in \mathscr{C}$ keeping either a ratio $r \in [0,1]$ of the cluster population, or a fixed specimen. At the end of the process, a set of sampled clusters $\hat{\mathscr{C}} = \bigcup_j \hat{C}_j$ is obtained. Let m be the number of elements to extract. In case of fixed ratio r, I set $m = \lceil r||C_j|| \rceil$, and then pick $\hat{C}_j = sample(C_j, m)$. In case of a fixed sampling, I choose m a priori, and I select elements as $\hat{C}_j = sample(C_j, m)$.[1]

$sample(C_j, m)$ is a function that extracts m samples from C_j. I consider three samplings:

- Random sampling: selecting m objects at random from the elements of C_j, i.e., $sample(C_j, m) = rand(C_j, m)$;
- Proportional sampling: selecting m objects at random from the elements of C_j where m is chosen according to the ratio $r \in [0,1]$, i.e., $sample(C_j, m) = rand(C_j, m)$, where $m = \lceil r||C_j|| \rceil$,
- Percentile sampling: selecting the elements that best represents the different kind of URLs present in a cluster, i.e., $sample(C_j, m) = percentile(C_j, m)$.

$percentile(C_j, m)$ extracts m representatives by looking at the distribution of mean distances for each URL $s_i \in C_j$

$$\left\{ E_{s_k \in C_j}[d_{URL}(s_i, s_k)], \forall s_i \in C_j \right\} \tag{4.1}$$

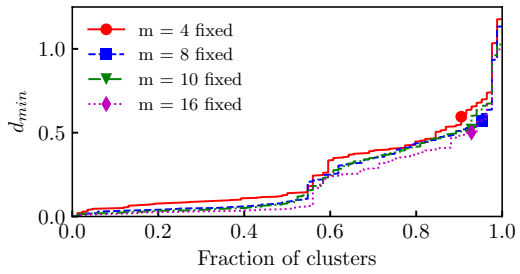
The selected elements are the ones that correspond to values that divide in equally sized sets the cluster, i.e., that correspond to the m percentiles. The idea behind percentile selection is having a set of cluster samples containing both elements that are in the center area of a cluster and the ones at its border. Note that in case of $m = 1$, $percentile(C_j, m)$ would select the so-called medoid, i.e., the element whose average dissimilarity to all the objects in the cluster is minimal.[2] The medoid is generally an appropriate choice to describe a group of elements, but it is more appropriate for spherical and homogeneous clusters. Since a cluster in IDBSCAN consists of a chain of interconnected smaller spherical dense areas, the choice of only one point would exclude other possibly distinguishing instances. In this sense, the percentile sampling produces a sampling that better represents the population of the cluster.

4.2.1 Sampling on Synthetic dataset

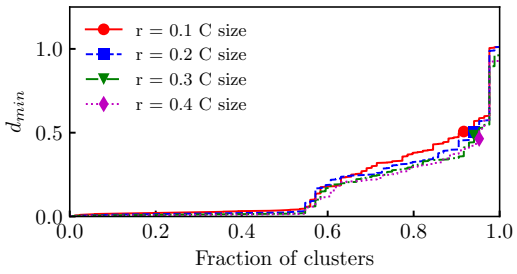
I compare the sampling methodologies over the results of the IDBSCAN clustering applied in the artificial dataset scenario 2.2.1, with 20 000 points, 100 clusters, and three-dimensional feature space.

[1] In case $|C_j| \leq m$, all elements are selected.

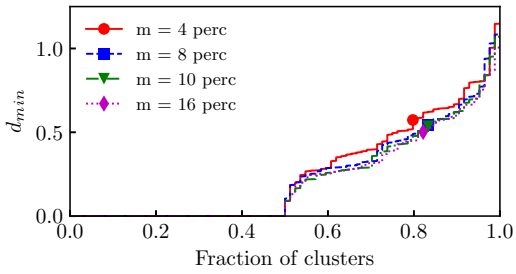
[2] The medoid is different from the centroid since the first consists in selecting an element among the ones of the cluster, while the second does not have this restriction.



(a) Fixed sampling approach.



(b) Proportional sampling approach.



(c) Percentile sampling approach.

Figure 4.1: Sampling applied to the artificial dataset with 100 clusters, 50 of which were already seen in the past.

I set up an experiment where starting from the data $\mathscr{D}$, I generate a subset $\boldsymbol{D} \subset \mathscr{D}$. Through $\boldsymbol{D}$ I build the System Knowledge $\mathscr{Z}(0)$, which contains the clusters generated by applying IDBSCAN on $\boldsymbol{D}$. From the complete set $\mathscr{D}$ I extract the ensemble of clusters $\mathscr{C}(1)$, which mimic the second day of analysis. I apply sampling and compare $\hat{\mathscr{C}}(1)$ to $\hat{\mathscr{Z}}(0)$. The expect result is that IDBSCAN would generate similar clusters for the common elements in the two sets and that then the System Knowledge algorithm would to identify half of the clusters as already known, and the other half as new.

In this experiment the first part contains points belonging to 50 labels, i.e., clusters, present in the *Synthetic-labeled Dataset*. The second set contains, instead, all points in *Synthetic-labeled Dataset*. I compute the clustering on the first set and extract the representatives. I then rerun the clustering with all points from all 100 clusters, extract the representatives, and compare them with the previous representatives. The idea is, as said before, to check if the clusters sampling technique allows the identification of the same clusters (the 50 clusters present in the first and second dataset), checking the similarity between the new representatives and the old ones. I repeat this experiment for the different sampling strategies and an increasing number of representatives.

Figure 4.1 shows the results. For each test, I report, for each cluster $\boldsymbol{C}$, the minimum distance between its sample representation $\hat{\boldsymbol{C}}_j$ and the System Knowledge $\hat{\mathscr{Z}}$, i.e.,

$d_{min}(\hat{C}_j, \hat{\mathcal{X}})$, sorted by increasing d_{min}. Ideally, I would expect to have $d_{min} = 0$ for those 50 clusters that are in common, while $d_{min} \geq \alpha$ for the 50 new clusters that are present in the second batch only.

Results show that the sampling and System Knowledge enhancement work quite well, with the percentile sampling performing better given its deterministic approach to select representatives. As I could expect, all of the three sampling methodologies perform better when increasing the number of representatives. The comparative approach seems to offer excellent results in this scenario. It is worth to recall that, in this benchmark, all clusters contain about 200 points each, so that also random sampling and fixed sampling present good results.

At last, considering the choice of the threshold α, I can see that any value higher than 0.1 would offer a good separation between the 50 old clusters, and the 50 new clusters.

4.2.2 Sampling with URLs

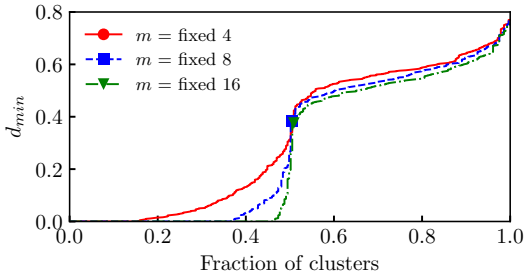
To choose which strategy works best in the real case scenario, I run a second experiment in which I again split the set of URLs in the first day of data 2.2.3 into two sets. I run the clustering considering the first half of URLs, extract representatives for each identified cluster, and add them to the System Knowledge. I then rerun the clustering considering all URLs, extract representatives, and match the new clusters with those in the System Knowledge. In this case, too, I expect about 50% of clusters to be old, i.e., $d_{min} < \alpha$, and 50% to be new, i.e., $d_{min} \geq \alpha$.

Results are depicted in the plots of Figure 4.2, which compares the three different sampling strategies, with different parameters. The figure shows that the System Knowledge matching works as expected. The more the number of representatives, the more the ideal step-curve-behavior is visible. The approximation is satisfactory, picking a fixed m equal to 16, and very similar to the step curve with proportional r of 20% or 30%.

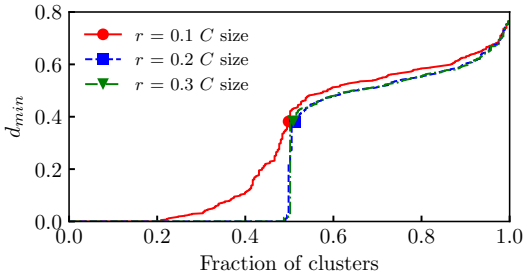
For both the experiments, I obtain the best results when using percentiles, whose smart sampling guarantees optimal results. Indeed, when I consider the percentile approach, I always obtained a perfect distance of 0 for the clusters that contain the same elements as the compared ones. That is happening because two sets are equal, and I deterministically select the representatives.

Considering the choice of α when actual URLs are considered, Figure 4.2a, Figure 4.2b and Figure 4.2c clearly show that the new clusters tend to be very dissimilar from the old ones, and that any $\alpha \in [0.2,0.4]$ is a proper choice. To not discard potentially new and interesting clusters, in the following, I choose a value of $\alpha = 0.3$.

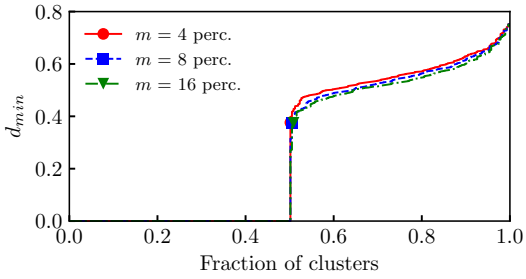
At last, enlarging the number of representatives has the drawback of increasing the computation complexity, due to the need to compute $O(m^2)$ $d_{URL}(.)$. Figure 4.3 shows the experimental computational time using lin/log scales considering the set of 60 000



(a) Fixed sampling approach.



(b) Proportional sampling approach.



(c) Percentile sampling approach.

Figure 4.2: d_{min} when 50% of traffic is the same and 50% is new. Different choices of sampling approaches.

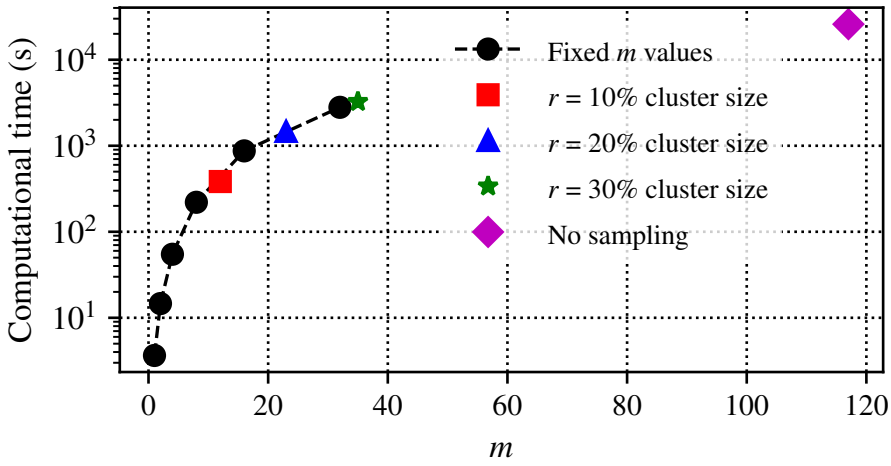


Figure 4.3: Computation time for different sampling strategies. Without sampling, the comparison of the System Knowledge would require too much time.

URLs. For variable fraction r, I report results in the plot assigning to m the value corresponding to the closest integer to the average number of elements in the clusters. As expected, the curve grows quadratically for m (logarithmically in log scale), with $m = 32$ and $r = 20\%$ (on average 23 samples) or 30% (avg. 35 samples) that already have a complexity larger than 3 000 s. Without sampling, these experiments would require more than 7 hours to complete. Considering the System Knowledge would have thousands of clusters and that this step shall be completed every ΔT, the best trade-off between cluster similarity identification and computational time is obtained using a fixed $m = 16$.[3]

4.3 System Knowledge Enhancement

The System Knowledge $\hat{\mathcal{Z}}(t)$ maintains the set of clusters found in the past. At the beginning $\hat{\mathcal{Z}}(0) = \emptyset$. Given a sampled cluster $\hat{C}_i$ I want to identify the closest cluster found in the past. Let

$$d_{min}(\hat{C}, \hat{\mathcal{Z}}) = \min_{\hat{Z} \in \hat{\mathcal{Z}}} \left(d\left(\hat{C}, \hat{Z}\right)\right)$$

$$\text{where } d(\hat{C}, \hat{Z}) = \min_{\substack{c \in \hat{C} \\ z \in \hat{Z}}} d_{URL}(c, z) \quad (4.2)$$

Let $\hat{\mathcal{C}}(t)$ be the result of the clustering of the current batch. I need to check if a cluster $\hat{C}_j(t) \in \hat{\mathcal{C}}(t)$ contains the similar content of an already registered one, or if it represents new traffic. For the cluster $\hat{C}_j(t)$, the most similar cluster $\hat{Z}_l(t-1) \in \hat{\mathcal{Z}}(t-1)$ is

$$\hat{Z}_l(t-1) = \arg\min \left(d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right)\right) \quad (4.3)$$

A cluster is then considered as new if the minimum distance is larger than the threshold α. The System Knowledge is updated as follows:

$$\hat{\mathcal{Z}}(t) = \hat{\mathcal{Z}}(t-1) \cup \left\{ \hat{C}_j(t) \in \hat{\mathcal{C}}(t) \mid d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right) \geq \alpha \right\} \quad (4.4)$$

That is, I add a new cluster found at time t if its distance to the closest cluster is higher than α.

4.3.1 In vitro experiment

To evaluate the reaction of the System Knowledge with respect to the appearance of anomalous elements, I design a controlled experiment in multiple stages. I start from an initial group $UG(0)$ of almost 33 000 unique URLs extracted at random from the 2.2.3 HTTP dataset. I then artificially create new groups $UG(1), UG(2)$, and $UG(3)$, where I progressively inject URLs belonging to different applications. I first add a block of 200 torrent URLs, i.e.,, $UG_1 = UG_0 \cup \{TorrentURLs\}$. Next, I add 228 malicious

[3] In this case, too, CPU time can be reduced by computing d_{URL} in parallel.

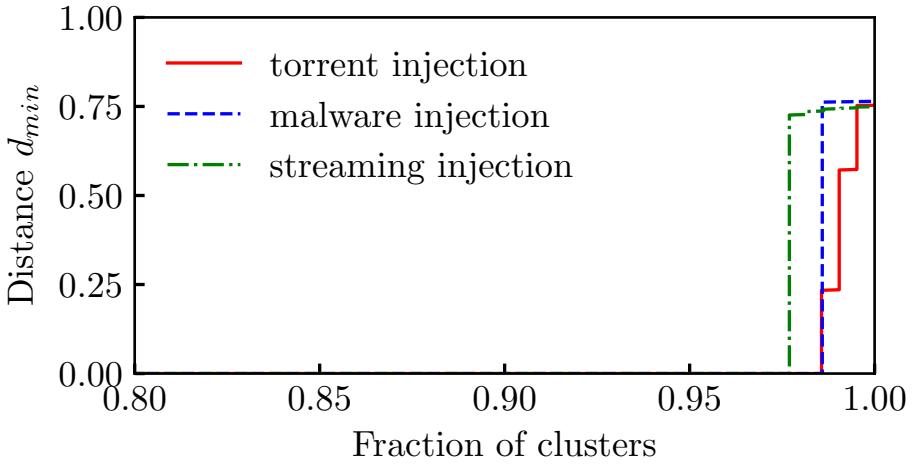


Figure 4.4: Curves of distances when new traffic is injected in the controlled experiment. Top 20% clusters are reported.

Table 4.1: New clusters after the comparison with the System Knowledge.

Experiment stage	**d_{min}**	**Main hostname(s)**
UG_1 Torrent	0.75	b-0.ad.bench.utorrent.com
	0.57	scorecardresearch.com, pixel.quantserve.com
	0.23	torrent.gresille.org
UG_2 Malware	0.76	wuptywcj.cn
	0.76	*-6nbcv.com, iau71nag001.com
	0.76	bangl24nj14.com, switcho81.com
UG_3 Streaming	0.75	198.38.116.148
	0.74	23.246.50.136, 198.38.116.148
	0.74	198.38.116.148
	0.73	23.246.50.136, 198.38.116.148
	0.72	198.38.116.148

URLs generated by hosts infected by *TidServ*, i.e.,, $UG(2) = UG(1) \cup \{TidservURLs\}$. Finally, I inject 549 URLs generated by a popular streaming service, i.e.,, $UG(3) = UG(2) \cup \{StreamingURLs\}$.

After each stage of clustering and comparison with System Knowledge, I check the ability to identify the new traffic. Results are reported in Figure 4.4, which shows the

Table 4.2: Behavior of the system during the week.

	Mar-01	Mar-02	Mar-03	Mar-04	Mar-05	Mar-06	Mar-07
Unique URL	59543	62842	67789	61849	77770	87928	88396
Daily Clusters	283	322	348	304	396	428	431
System knowledge	283	475	643	765	927	1097	1267
System enhancement	283	192	168	122	162	170	170

minimum distance $d_{min}(\hat{C}(t), \mathcal{Z}(t-1))$ between clusters found in $UG(t)$ and those in the System Knowledge build on previous steps. I report only the first 20% of clusters, ordered by distance. As clearly shown, LENTA is able to recognize the new traffic: first, d_{min} is equal to zero for those clusters in $UG(t)$ that were already present in $UG(t-1)$. Second, and more importantly, the new traffic is clustered in totally different clusters, whose d_{min} is much higher than $\alpha = 0.3$.

In detail, Table 4.1 depicts the results of the experiment. First, all clusters contain only new URLs injected in each step of the process. Second, notice that the system identifies multiple new clusters for each stage. This behavior is welcome since each cluster corresponds to a semantically different service. For instance, for the video streaming case, each cluster corresponds to videos served for different platforms (iOS, Android, and PC), and torrent clusters correspond to different swarms and trackers. Third, $d_{min} > 0.3$ for all clusters but one in the Torrent data, for which $d_{min} = 0.23$. This cluster would be associated with a previously seen cluster. The association is correct, and URLs have a very similar syntax to the one already found and related to a tracker service, `tntvillage`.

4.4 Ageing

When $d_{min}(\hat{C}_j(t), \mathcal{Z}(t-1)) < \alpha$, two clusters are considered similar, so they contain the same kind of information. The new cluster is associated with the old one and may contain new knowledge, e.g.,, some important changes in the particular service or differences in the structure or information carried by URLs. It is vital to register, if possible, those updates.

I apply a random replacement policy. That is, I substitute each element $z_i \in \hat{Z}_l(t-1)$ with the element $c_i \in \hat{C}_j(t)$ with a certain probability p. So,

$$z_i := c_i \leftarrow p \;\; \forall i \in [1, m], \;\; z_i \in \hat{Z}_l(t-1), c_i \in \hat{C}_j(t) \tag{4.5}$$

In doing so, we update the System Knowledge representatives, ageing and replacing "old" ones with fresher information.

4.5 Pruning Based on Inactivity

The System Knowledge keeps information on clusters since the first time they had been added to it. Let t_{init} be the first time in which a cluster appears in $\hat{Z}$, and t_{last} be the last time it has been encountered, i.e., when the System Knowledge associates, through similarity, a newly found cluster to it.

The information about the "last seen" date is crucial to understand the development of network traffic clusters. Knowing which are the most recently visited clusters, it is possible to understand which are the trends in the network. At the same time, looking at the "old" clusters helps in notifying changes in behavior, in suggesting the rise of new services or the replacement of old ones.

On the other hand, keeping old, clusters that are not contacted anymore, can represent a limitation in terms of storage and a bottleneck during the analysis, since each time the system compares the sample of the newly found clusters with all the elements present in the System Knowledge. For this reason, it is essential to implement a pruning mechanism to remove those inactive clusters that contain content that users did not visit in the recent past. The information of the last active date, i.e., t_{last}, is leveraged to remove old and inactive clusters. Given $\hat{Z}$, it is possible to define $\Delta T_{inactive} = t - t_{last}$. If $\Delta T_{inactive} \geq \Delta I_{thresh}$, with ΔI_{thresh} defined as *Inactivity Threshold*, then $\hat{Z}_{inactive}$ is removed from $\hat{\mathcal{Z}}$. Section 5.1 contains a discussion of the impact of ΔI_{thresh} in a real-case scenario.

Chapter 5

Application

This chapter contains the verification, in application scenarios, of the system composed of the components described in Chapters 3 and 4, IDBSCAN clustering, and System Knowledge, i.e., LENTA. The overall system is analyzed in real scenarios, prolonged over time, containing the navigation activities of different users. This thesis explores two scenarios.

Section 5.1 refers to 3 weeks of analysis on passive traces, i.e., *HTTP-not-labeled Dataset*. The test reported in this section analyzes the System Knowledge over daily bins of unrepeated URLs extracted from all users. In this way, the clusters possibly contain pages and resources visited by more than one user. This approach allows an overall analysis of the whole network.

In Section 5.2, the dataset contains HTTP and HTTPS webpages, intercepted from the users' machine. The analysis here is per-user, i.e., it implies the performing of IDBSCAN clustering for each user. The investigation objective is two-fold. First of all, it aims at observing the single-day and evolutionary activity of single users. Secondly, it focuses on scrutinizing the commonalities in the actions of the monitored group.

The results reported in this chapter are part of the work presented in [56].

5.1 HTTP Traffic Over Time

Figure 5.1 reports the process of System Knowledge evolution over the 21 days under analysis. In the first day all the clusters are added, thus being all labeled as new (in green). From the second day, we can notice different operations: (i) some new clusters join the System Knowledge, (ii) the system operates updates on others when it finds similar clusters (blue), and (iii) the remaining group (gray) is in idle. The growth of $\hat{\mathcal{F}}$ continues up to the beginning of the second week when the pruning action - based on inactivity - starts evicting some old, inactive clusters (yellow). The red bar details how many discarded clusters reappear in the future steps.

The values obtained for the last two features depend on the size of ΔI_{thresh}. To check

the impact of this choice, Figure 5.2 reports the number of clusters that would be evicted from and eventually re-inserted into system knowledge. As expected the number of both deleted and reappearing clusters decreases enlarging ΔI_{thresh}. This behavior indicates a non-negligible periodicity, especially in consecutive days, that may suggest a natural stabilization of the system knowledge size over long periods. For extended analysis, a higher ΔI_{thresh} may be more suitable, following memory and storage limitations, to let the system have time to level off.

Compare Figure 5.1 with the growth in Fig 2.2. The number of unique URLs tops more than 420 000 after seven days, with on average 72 000 unique URLs per day. The number of clusters instead solely reaches 1 600, with less than 200 newly found clusters per day. In a nutshell, this approach can decrease the amount of information the security analysts have to process by three orders of magnitude so that they have to inspect less than 200 clusters per day instead of managing several tens of thousands of unique URLs.

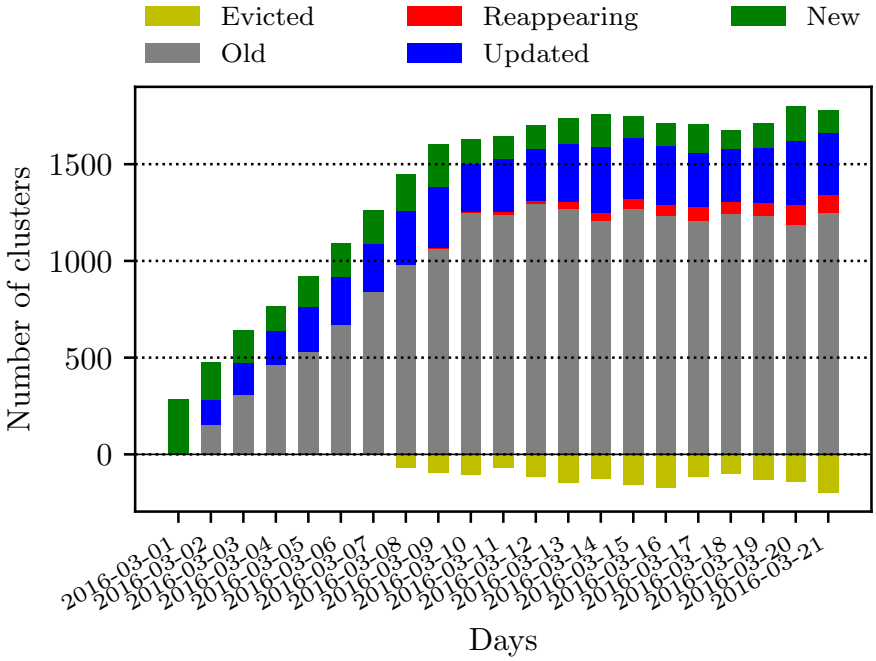


Figure 5.1: Daily enhancement of system knowledge for HTTP data.

The variability of URLs grouped in the same cluster also simplifies the investigation of the involved services. For instance, I checked some clusters that came into sight after each System Knowledge enhancement phase. I report, for each day of the first week of observation, the five new most different clusters for the previously collected traffic, i.e., those for which $d_{min}(C_j(t), \hat{\mathcal{Z}}(t-1))$ is highest.

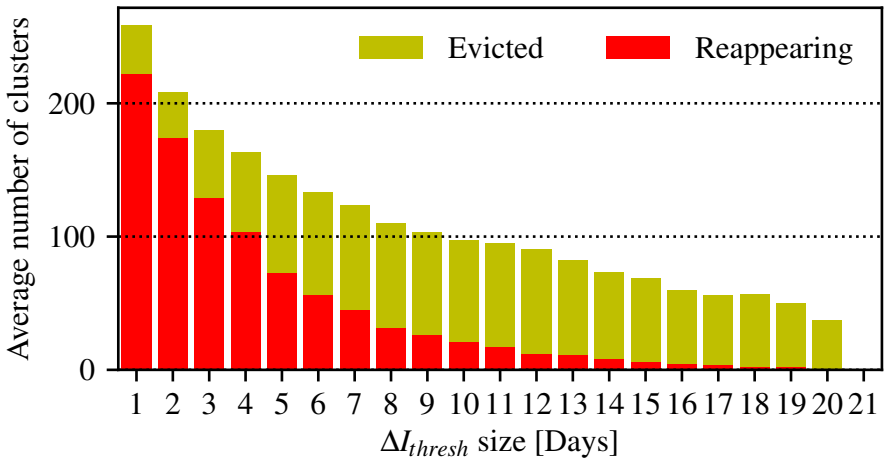


Figure 5.2: Number of evicted and reappearing clusters given different ΔI_{thresh}.

Table 5.1: Most interesting clusters obtained by the daily comparison with the system knowledge in the controlled experiment.

Day	Main hostname (unique hostnames)	Activity	Day	Main hostname (unique hostnames)	Activity
	adnxs.com (3)	Advertising		ams1.mobile.adnxs.com (1)	Advertising
	www.bing.com (1)	Search Engine		ads1-adnow.com (3)	Advertising
Mar-02	amazon.it (3)	E-commerce	Mar-03	uk-ads.openx.net (1)	Advertising
	doubleverify.com (9)	Advertising		c.3g.163.com	Chinese Website
	mp.weixin.qq.com (1)	Chinese Website		googleapis.com (1)	Cloud Storage
	banzai-d.openx.net (1)	Advertising		engine.bitmedianetwork.com (1)	uTorrent Adv
	dt.adsafeprotected.com (1)	**Hijacker**		**62.210.188.202:8777** (1)	**Suspicious Port**
Mar-04	**gvt1.com** (3)	**Hijacker**	Mar-05	**adaptv.advertising.com** (1)	**Suspicious Adv**
	windowsphone.com (1)	CDN Marketplace		pubnub.com (16)	Messaging
	ocsp.digicert.com (1)	Certificate inspection		**irs01.com** (1)	**Suspicious Tracking**
	23.246.50.130 (5)	Netflix Italy		aww.com.au (2)	News
	198.38.116.148 (3)	Netflix Germany		*.liverail.com (1)	Advertising
Mar-06	23.246.50.136 (3)	Netflix Italy	Mar-07	**spaces.slimspots.com** (1)	**Adware attack**
	23.246.51.136 (2)	Netflix Italy		googleusercontent.com (2)	Page Translation
	178.18.31.55:8081 (7)	**Suspicious Streaming**		**s8.algovid.com** (1)	**Malicious Adv**

Table 5.1 details the results. In this case, as well, the services are related to streaming, advertising, and e-commerce services. Some unexpected traffic emerges as well; for instance, on March 3rd, the `c.3g.163.com` cluster emerges. It is related to the Chinese web portal `www.163.com`, a service not reported in the previous days. URLs are related to a newsfeed specific service. March 4th and 5th, I register some suspicious or malicious traffic. Clusters are related to browser hijacker services, mainly adware, and aggressive advertisement. March 6th is distinctively captivating. Eight out of ten most different clusters contain URLs characterized by IP addresses which resolve Netflix Italy or Netflix Germany CDNs. These were not found in the previous days, highlighting a change in the Netflix load balancing policies. The other cluster contains traffic from `178.18.31.55:8081`, connected to *liverepeater*, a keyword related to illegal

streaming content. Finally, in the last day, some suspicious traffic is visible: a rare service like `aww.com.au`, an Australian news website, and webpages translated using the Google Translate online service (curiously translating an adult-content website, possibly to evade content filtering policies).

5.2 HTTPS Traffic Over Time

Here I report on my second experiment with HTTPS traffic collected via the ERMES proxy, see Section 2.2.4. The users were monitored and stimulated in producing traffic for 30 days, as more continuatively as possible. The collection in exam considers the users active for two consecutive weeks, in order to explore their activity day after day, using LENTA. Indeed, differently from the previous analysis, here the exploration considers individual users (albeit after anonymization). Starting from the traffic generated each day by each volunteer, the process aims at first to characterize the usage patterns of each user, and secondly, to observe how it changes over time. Plus, LENTA has been specifically configured in order to collect the information of which and how many users visited a cluster present in the System Knowledge, to inspect the most prominent behaviors.

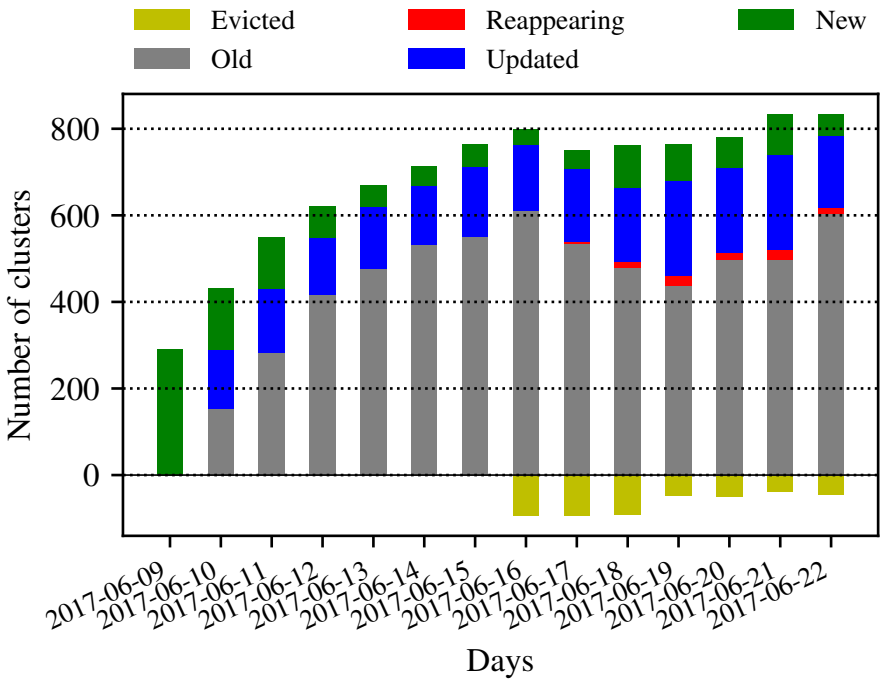


Figure 5.3: Daily enhancement of system knowledge in HTTPS traces.

Figure 5.3 shows the behavior of the System Knowledge for the current investigation. The functioning of the overall system is comparable to the preceding experiment, reported in Figure 5.1. However, in this use case, the results produce information about the behavior of users, showing the flexibility of the system in supporting data exploration. In relation to the smaller sample of users, here the per-user exploration produce in proportion a larger number of clusters. This is due both to the fact that I perform the clustering on a per-user basis, and thanks to the availability of HTTPS traffic.

In the work of this thesis, I focus on the analysis of the most common categories of services accessed by users, in order to analyze the behavior of users and the impact of trackers in the amount of traffic inspected from the users activity. The focus is on 171 clusters that resulted in being common for at least two users. The categories extraction derives from manual inspection of clusters, in conjunction with the support of public lists for data categorization. This step results greatly simplified thanks to the similarity and expressiveness of URLs contained in each cluster. Overall, I identified 20 coarse categories of services.

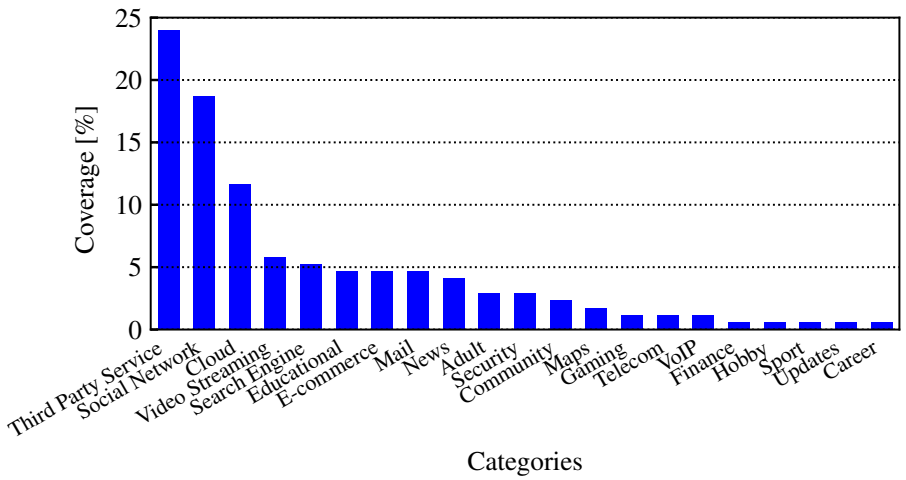


Figure 5.4: Most popular categories extracted from the clusters visited by at least two users.

Figure 5.4 reports the fraction of clusters that fall into the same category. Almost a quarter of the groups are related to third-party services, which include advertisement, web tracking, and analytics; their pervasiveness affects the results of the system. Not surprisingly, social networks occupy the second position. In the third place, I can find cloud services belonging to Google and some CDN used for image storage. Overall, I can map URLs to categories easily. The system dramatically simplifies manual labeling, thanks to the rich information offered by URLs in each cluster.

An extract of the clustering results, together with its characterization, is publicly

available on the SmartData@PoliTo website. [1]

[1] https://smartdata.polito.it/lenta-dataset/

Chapter 6

Conclusions and Future Research Directions

6.1 Summary and Contribution

Today, the Internet environment is growing in size and complexity. For this reason, researchers, Internet Service Providers (ISPs) and companies are working on new development for network traffic monitoring and characterization. This is a critical task for network administration and analysis. These approaches can help in identifying patterns and trends, detecting malicious behaviors, simplifying the examination, supporting the analysts in analyzing network traffic.

In this direction, in this work, I presented a methodology for the fast identification of HTTP/HTTPS-based services by looking at URL strings similarities. This approach avoids using other features, and does not require any labeling. The results showed how this methodology reduces the amount of traffic that needs a manual check and eases the observation of changes in the network traffic. It exposes well-formed clusters of URLs, which significantly simplifies the identification of possibly malicious and undesired traffic (Chapter 3).

I designed a recursive version of a clustering algorithm over daily HTTP/HTTPS traffic generated by hosts in a network, called IDBSCAN. I tested it against other off-the-shelf algorithms, namely DBSCAN, HDBSCAN, CANF, and OPTICS, using both a synthetic dataset and a real use case with URL objects, showing the benefits of my proposed solution (Chapter 3). The code is publicly available on GitHub. [1]

I outlined an evolutionary system for the temporal analysis of URL clustering. The design includes the engineering of the storage and comparison solution, a sampling methodology for the clusters, and ageing and pruning techniques to keep the system

[1] https://github.com/AndreaMorichetta/compute_clustering/blob/master/compute_clustering.py

up-to-date (Chapter 4). The evolutionary approach has been applied using two different observation points, a passive probe for HTTP traffic, and a MITM proxy installed on users' machines. The results show that the overall methodology, applied in a long-term observation, can identify anomalies in the traffic and changes in users' behavior (Chapter 5).

The findings in this work can be useful in different ways. They have a role in reducing the problem complexity, quickly producing an outcome for the analyst to whom are offered few hundreds of clusters instead of several hundreds of thousands of URLs. Thanks to the use of an unsupervised methodology, there is no need for labels, often a cumbersome problem, thus facilitating exploratory approaches. The improved clustering algorithm does not require the setting of parameters through complex and long tuning processes. This approach can be useful not only for network analysts but, in general, for researchers or companies interested in this kind of solution.

The evolutionary, general-purpose methodology allows a continuous analysis in time, facilitating traffic investigation, the extraction of patterns, anomaly detection, and encourage progressive traffic modeling.

6.2 Future Work

An extra effort is necessary to extend big data approaches to all the stages of the system to better scale the analysis. The work can go from the study of more efficient and scalable clustering techniques, which still has to be computed locally.

Other types of distances may be used in order to speed up the step, which until now represent the main bottleneck in its execution. A more organic collaboration of the different system components is also essential to improve stability. Making the code and the tools more uniform, as well as extending the number of data typologies that the system can handle, will help in broadening the application scenarios.

Regarding the last point, some of these contexts can be other use cases that involve lexical features, e.g., hostnames in DNS queries, or user-agents in HTTP requests, or completely different applications, e.g., topics evolution in scientific research articles.

In addition to my dissertation research, in the last months, I had the chance to focus on a new and fascinating subject that of conceiving methods and techniques to explain better the functioning and the decisions provided by machine learning models, such that human experts can understand these. This problem is better known as eXplainable AI (XAI).

Having humans in the loop of machine learning, especially in critical applications like network security, where the accuracy and reliability of the final decision are crucial, is of essential importance. A future effort can be the study and the implementation of collaborative and explanatory evaluation methodologies. In particular, they could be combined with other information, like external sources, e.g., blacklists, categories list, or other parameter from the traffic log, e.g., IPs, user-agents or referrer, in order to

better assist the decision. This approach could be important in case of lack of detailed information, because of encrypted traffic.

Thanks to the research developed in my Ph.D. I matured strong knowledge in unsupervised learning, network traffic analysis, and network security, as well as big data technologies. Working on these topics allowed me to deepen specific themes as well as challenge myself with different problems and applications.

Appendix A

Steps Towards Explainable AI

The work reported here is part of the article [53] that will be presented at the 3rd ACM CoNEXT Workshop Big-DAMA, in December 2019.

A.1 Introduction

The application of unsupervised learning techniques, and in particular clustering, offers invaluable help in the analysis of network measurements to discover underlying characteristics, group similar elements together, and identify eventual patterns of interest. Unfortunately, clustering does not always provide precise and clear insight into the produced output, especially when the input data structure and distribution are not clear. The following sections will present EXPLAIN-IT, a methodology which deals with unlabeled data, creates meaningful clusters and suggests an explanation of the results to the end-user. EXPLAIN-IT relies on a novel explainable AI approach, which allows understanding the reasons leading to a particular decision of a supervised learning-based model, but extending its application to the unsupervised learning domain. The steps toward explainability in unsupervised solutions are reported applying EXPLAIN-IT to the problem of YouTube video quality classification under encrypted traffic scenarios.

A.2 System Description

The goal of EXPLAIN-IT is to explain the outcome of unsupervised algorithms. In general terms, it addresses the process of knowledge discovery in datasets, providing a comprehensive tool tackling the variety of steps of this process. Figure A.1 depicts an overview of the system and its multiple steps. EXPLAIN-IT consists of two consecutive analysis steps, namely (i) reduction of the cardinality of the problem under analysis (e.g., data summarization and compression), and (ii) knowledge extraction and building of explanations.

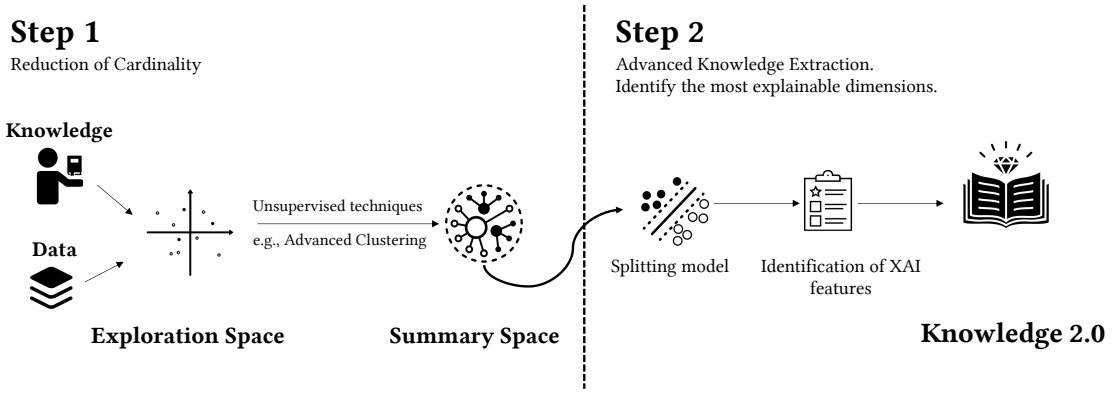


Figure A.1: The EXPLAIN-IT system. Data is firstly embedded into the *exploration space*, relying on expert knowledge when available. The *summary space* is the result obtained by clustering the exploration space. Next, it builds a supervised data splitting model out of the clustering results. Finally, it applies an XAI approach (LIME) to this splitting model, interpreting the contents of the clusters by adding local interpretations.

Exploration and Summary Spaces

In the first step, EXPLAIN-IT relies (when available) on the knowledge of experts to extract the right features, structuring the data analysis and the way for interpreting the results. This process may include feature selection and engineering methodologies. The definition/extraction of features corresponds to embedding the data into an *exploration space*, which serves as a basis for the data exploration process. Naturally, when no expert domain knowledge is available, the embedding would take place in an entirely blind fashion, using the features defined by the analyst. Once the data is embedded, EXPLAIN-IT uses unsupervised learning techniques to explore it, looking for relevant structures of interest. EXPLAIN-IT uses in particular clustering techniques, but any unsupervised methodology which creates a summary of the data can be applied. Clustering here plays the role of a meta-learning approach, which reduces the complexity of the analysis by aggregating similar instances, generating what I refer to as the *summary space*. Different clustering algorithms can be used, depending on the desired result, amount of data, space dimensionality, and other constraints. For example, in case the user wants to obtain a certain number of groupings or classes, solutions that require the number of clusters as input are preferred, e.g., K-Means. Otherwise, if the data structure itself should determine the exploration process, or if anomaly detection is relevant in the process, other techniques such as density-based ones (e.g., DBSCAN) are better suited.

Modeling the Summary Space

The second step of EXPLAIN-IT consists of automatically characterizing the resulting clusters obtained in the summary space. A common way to measure the effectiveness of clustering algorithms [36], is to use intrinsic or derived characteristics, like complexity, steadiness, computational time, as well as internal and external validity metrics, like the silhouette and the rank indexes. Those measures are useful in evaluating the goodness of the algorithm. However, they provide little insight into what clusters contain. The main idea is, therefore, to identify, for each of the obtained clusters, the most relevant features explaining the assignment of each data instance to it. While this could be in-principle done by per-feature analysis and using weighted distances and linear discriminant functions, there is always a limitation of such an approach, based precisely on the considered notion of distance. Depending on which type of distance metric one would use, the obtained explanations would be different. For example, Euclidean distances would favor features defining spherical-like rules explaining the results, whereas sparse spaces and heavy-tailed distributions would impact probability-based or correlation-based distances. Also, global linear discrimination would not perform accurately in most practical cases. Other empirical approaches, such as manual inspection, relevant features description, or extraction of the most representative data instances, suffer from scalability, and the limitations mentioned above.

In the absence of a general solution to this problem, and inspired by the notions behind white-box and black-box XAI, I rely on a purely data-driven approach and decide to model the results of the clustering step through a supervised learning model, which I then explain through XAI. As I said before, to improve discrimination power, I do not rely on natively interpretable models (e.g., decision trees or linear discriminant functions), but I consider more powerful data splitting models. Here, in particular, I take Support Vector Machine (SVM) discriminant models [73], which are well-known for their performance to identify non-linear and complex boundaries among instances, relying on the use of kernel functions and the so-called "kernel trick" to construct such boundaries.

XAI with LIME

The final step of the analysis consists of using black-box XAI approaches to finally interpret the structure of the summary space by explaining the SVM-based modeled clusters. In particular, I use LIME [65], a model-agnostic interpretation approach which relies on local model linearization to identify the most relevant features leading to a particular decision, for an individual data instance. LIME relies on sampling for local model exploration and linearization. To explain the LIME approach, let us assume I have a complex model $f(\cdot)$, and a specific instance x for which I want to explain the features leading to $f(x)$. LIME constructs a natively interpretable model $g(\cdot)$, which is *locally faithful* to $f(\cdot)$ in the vicinity of x, the latter captured by certain similarity measure $D_x(\cdot)$. For doing so, LIME randomly generates new instances z around x, which are

then weighted by $D_x(z)$ to define their local relevance. Finally, $g(\cdot)$ is built based on inputs z and their corresponding labels $f(z)$. In particular, LIME uses linear discriminant functions to build $g(\cdot)$.

Naturally, other XAI methodologies such as SHAP [45] are also very promising and could be part of EXPLAIN-IT. SHAP has solid mathematical foundations, but it has a much higher complexity that makes it hard to use on exploratory approaches. However, it results in better explanations - or at least approximations, making it more reliable in case of more stringent requirements. Nevertheless, LIME offers the best trade-off between computational time and interpretability, therefore its high popularity.

By combining the explanations provided by LIME for each data instance belonging to each cluster, EXPLAIN-IT finally provides guidelines easing the interpretation of the clustering results. The overall solution is very modular, allowing different settings and flexibility in the various stages of the process.

Appendix B

Publications, Awards, Patents and Collaborations

In this appendix, I report the list of the published papers, as well as prizes. I also present my collaborations, and other activities connected to my research career.[1]

Journals:

1. D'alconzo, Alessandro; Drago, Idilio; Morichetta, Andrea; Mellia, Marco; Casas, Pedro, A Survey on Big Data for Network Traffic Monitoring and Analysis, in: IEEE Transaction on Network and Service Management, 2019
2. Morichetta, Andrea; Mellia, Marco, Clustering and evolutionary approach for longitudinal web traffic analysis, in: Performance Evaluation, 2019
3. Morichetta, Andrea; Mellia, Marco, LENTA: Longitudinal Exploration for Network Traffic Analysis from Passive Data, in: IEEE Transaction on Network and Service Management, 2019

Conferences:

1. Morichetta, Andrea; Trevisan, Martino; Vassio, Luca, Characterizing Web Pornography Consumption from Passive Measurements, in: International Conference on Passive and Active Network Measurement, 2019
2. Faroughi, Azadeh; Javidan, Reza; Mellia, Marco; Morichetta, Andrea; Soro, Francesca; Trevisan, Martino, Achieving Horizontal Scalability in Density-based Clustering for URLs, in: Proceedings of the 2018 IEEE International Conference on Big Data, 2018

[1] More information can be found at https://www.telematica.polito.it/member/andrea-morichetta/.

3. Morichetta, Andrea; Mellia, Marco, LENTA: Longitudinal Exploration for Network Traffic Analysis, in: 30th International Teletraffic Congress (ITC 30), 2018

4. Ciociola, Alessandro; Cocca, Michele; Giordano, Danilo; Mellia, Marco; Morichetta, Andrea; Putina, Andrian; Salutari, Flavia, UMAP: Urban Mobility Analysis Platform to Harvest Car Sharing Data, in: Proceedings of the IEEE Conference on Smart City Innovations, 2017

5. Morichetta, Andrea; Bocchi, Enrico; Metwalley, Hassan; Mellia, Marco, CLUE: Clustering for Mining Web URLs, in: International Teletraffic Congress, 2016

Patents:

1. Mellia, M; Metwalley, H; Bocchi, E.; Morichetta, A., "METODO PER L'ESPLORAZIONE DI TRACCE PASSIVE DI TRAFFICO E RAGGRUPPAMENTO DI URL SIMILI", 2016

Research cooperations with other universities and research centers:

- Visiting student at the Austrian Institute of Technology (Vienna, AT) – Researcher. from January 2019 to July 2019. The collaboration focused on eXplainable AI (XAI) for unsupervised machine learning.

- Cisco Labs in San Jose, CA, In July 2017. Focus on Anomaly Detection and Security for DNS traffic analysis.

Travel grants and Prizes:

- TMA 2019 - Ph.D. School. Best Poster Award - Runner up, "Steps Towards Explainable AI for Clustering: the Case of Unsupervised QoE Analysis for YouTube Encrypted Traffic."

- Travel Grant TMA 2019.

- ITC30, 2018. Best Student Paper Award: "LENTA: Longitudinal Exploration for Network Traffic Analysis."

- Travel Grant TMA 2018.

- Travel Grant TMA 2017.

Other topics of interest:

- 2018 [Workshop Assistance] Assistance during the course "From Packets to Knowledge: Applying Data Science Approaches to Large Scale Passive Measurements" (prof. Idilio Drago) as part of the 2018 Traffic Monitoring and Analysis Ph.D. School
- 2017 [Lab Assistance] Lab Assistance and exercises preparation in ICT in Transport Systems (prof. Marco Mellia)
- 2016 [Talk and Assistance] Project description and assistance during Big Dive, 5th edition on behalf of SmartData@PoliTO center
- 2016 [Talk] Seminar talk in "Big Data Summer School" organized by "Ordine degli Ingegneri di Asti."

Bibliography

[1] *2019 Internet Security Threat Report*. https://www.symantec.com/security-center/threat-report. Accessed: 2019-10-26.

[2] Charu C. Aggarwal and Chandan K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman and Hall/CRC, 2013.

[3] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "A survey of network anomaly detection techniques". In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31.

[4] Marco Aldinucci et al. "HPC4AI: An AI-on-demand Federated Platform Endeavour". In: *Proceedings of the 15th ACM International Conference on Computing Frontiers*. CF '18. Ischia, Italy: ACM, 2018, pp. 279–286. ISBN: 978-1-4503-5761-6. DOI: 10.1145/3203217.3205340. URL: http://doi.acm.org/10.1145/3203217.3205340.

[5] Blake Anderson. *Hiding in Plain Sight: Malware's Use of TLS and Encryption*. https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption. Accessed: 2017-10-15.

[6] Mihael Ankerst et al. "OPTICS: ordering points to identify the clustering structure". In: *ACM Sigmod record*. Vol. 28. 2. ACM. 1999, pp. 49–60.

[7] Manos Antonakakis et al. "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware." In: *USENIX security symposium*. Vol. 12. 2012.

[8] Daniele Apiletti et al. "SeLINA: A Self-Learning Insightful Network Analyzer". In: *IEEE Transactions on Network and Service Management* 13.3 (Sept. 2016), pp. 696–710. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2597443.

[9] *Backdoor.Tidserv*. https://www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99. Accessed: 2015-01-20.

[10] Monowar H. Bhuyan, Dhruba K. Bhattacharyya, and Jugal K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools". In: *Commun. Surveys Tuts.* 16.1 (2014), pp. 303–336.

[11] Leyla Bilge et al. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." In: *Ndss*. 2011.

[12] Raouf Boutaba et al. “A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities”. In: *J. Internet Serv. Appl.* 9.16 (2018).

[13] Andrei Z. Broder et al. “Syntactic clustering of the Web”. In: *Computer Networks and ISDN Systems* 29 (1997), pp. 1157–1166.

[14] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. “Density-based clustering based on hierarchical density estimates”. In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2013, pp. 160–172.

[15] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. “A comparison of string distance metrics for name-matching tasks”. In: *IJCAI-03 Workshop on Information Integration*. 2003, pp. 73–78.

[16] Irving Cordova and Teng-Sheng Moh. “Dbscan on resilient distributed datasets”. In: *High Performance Computing & Simulation (HPCS), 2015 International Conf. on*. IEEE. 2015, pp. 531–540.

[17] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. “Clustering web pages based on their structure”. In: *Elsevier Data & Knowledge Engineering* 54.3 (2005), pp. 279–299.

[18] *Digital 2019: Global Digital Overview*. https://datareportal.com/reports/digital-2019-global-digital-overview. Accessed: 2019-10-26.

[19] *Digital 2019: Q4 Global Digital Statshot*. https://datareportal.com/reports/digital-2019-q4-global-digital-statshot. Accessed: 2019-10-26.

[20] Ergin Elmacioglu et al. “Psnus: Web people name disambiguation by simple clustering with rich features”. In: *International Workshop on Semantic Evaluations*. 2007, pp. 268–271.

[21] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. “Traffic classification using clustering algorithms”. In: *Proceedings of the 2006 SIGCOMM workshop on Mining network data*. ACM. 2006, pp. 281–286.

[22] Azadeh Faroughi and Reza Javidan. “CANF: Clustering and anomaly detection method using nearest and farthest neighbor”. In: *Future Generation Computer Systems* 89 (2018), pp. 166–177. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.06.031. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17325128.

[23] Azadeh Faroughi et al. “Achieving Horizontal Scalability in Density-based Clustering for URLs”. In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 3841–3846.

[24] Junhao Gan and Yufei Tao. "DBSCAN revisited: mis-claim, un-fixability, and approximation". In: *Proceedings of the 2015 ACM SIGMOD International Conf. on Management of Data*. ACM. 2015, pp. 519–530.

[25] Hongyu Gao et al. "Detecting and Characterizing Social Spam Campaigns". In: *ACM IMC*. 2010.

[26] Pedro García-Teodoro et al. "Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges". In: *Comput. Secur.* 28 (2009), pp. 18–28.

[27] Danilo Giordano et al. "YouLighter: A cognitive approach to unveil YouTube CDN and changes". In: *IEEE Transactions on Cognitive Communications and Networking* 1.2 (2015), pp. 161–174.

[28] A Ardeshir Goshtasby. *Similarity and dissimilarity measures*. Springer, 2012, pp. 7–66.

[29] Guofei Gu, Junjie Zhang, and Wenke Lee. "BotSniffer: Detecting botnet command and control channels in network traffic". In: (2008).

[30] Guofei Gu et al. "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection". In: SS'08 (2008), pp. 139–154. URL: http://dl.acm.org/citation.cfm?id=1496711.1496721.

[31] Dianwei Han et al. "A novel scalable DBSCAN algorithm with spark". In: *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE. 2016, pp. 1393–1402.

[32] John A. Hartigan. *Clustering Algorithms*. New York: Wiley, 1975.

[33] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. "Scalable Techniques for Clustering the Web". In: *International Workshop on the Web and Databases*. 2000.

[34] Fang Huang et al. "Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the Spark platform". In: *Remote Sensing* 9.12 (2017), p. 1301.

[35] Mubashar Hussain et al. "Towards ontology-based multilingual URL filtering: a big data problem". In: *The Journal of Supercomputing* 74.10 (2018), pp. 5003–5021.

[36] Felix Iglesias, Tanja Zseby, and Arthur Zimek. "Absolute Cluster Validity". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), pp. 1–1. DOI: 10.1109/TPAMI.2019.2912970.

[37] Ali Safari Khatouni et al. "Privacy issues of ISPs in the modern web". In: *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2017 8th IEEE Annual*. IEEE. 2017, pp. 588–594.

[38] Nizar Kheir et al. "Automated classification of C&C connections through malware URL clustering". In: *IFIP International Information Security Conference*. Springer. 2015, pp. 252–266.

[39] James F. Kurose and Keith W. Ross. *Computer networking: a top-down approach*. Addison Wesley, 2011.

[40] Yeonhee Lee and Youngseok Lee. "Toward Scalable Internet Traffic Measurement and Analysis with Hadoop". In: *SIGCOMM Comput. Commun. Rev.* 43.1 (2013), pp. 5–13.

[41] V.I. Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals." In: *Soviet physics doklady*. 1966, pp. 10–707.

[42] Hung-Jen Liao et al. "Intrusion Detection System: A Comprehensive Review". In: *J. Netw. Comput. Appl.* 36.1 (2013), pp. 16–24.

[43] Yingqiu Liu, Wei Li, and Yun-Chun Li. "Network traffic classification using k-means clustering". In: *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*. IEEE. 2007, pp. 360–365.

[44] Alessandro Lulli et al. "NG-DBSCAN: scalable density-based clustering for arbitrary data". In: *Proc. of the VLDB Endowment* 10.3 (2016), pp. 157–168.

[45] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[46] Guangchun Luo et al. "A parallel dbscan algorithm based on spark". In: *Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conf. on*. IEEE. 2016, pp. 548–553.

[47] Samuel Marchal et al. "A Big Data Architecture for Large Scale Security Monitoring". In: Proc. of the BigData.Congress. 2014, pp. 56–63.

[48] Max-Emanuel Maurer and Lukas Hofer. "Sophisticated Phishers Make More Spelling Mistakes: Using URL Similarity against Phishing". In: *Springer Cyberspace Safety and Security*. 2012.

[49] Andrew McAfee et al. "Big data: the management revolution". In: *Harvard business review* 90.10 (2012), pp. 60–68.

[50] Leland McInnes and John Healy. "Accelerated Hierarchical Density Based Clustering". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. Nov. 2017, pp. 33–42. DOI: 10.1109/ICDMW.2017.12.

[51] Hassan Metwalley, Stefano Traverso, and Marco Mellia. "Using passive measurements to demystify online trackers". In: *Computer* 49.3 (2016), pp. 50–55.

[52] Tom M. Mitchell. *Machine learning*. McGraw Hill, 1997.

[53] Andrea Morichetta, Pedro Casas, and Marco Mellia. "EXPLAIN-IT: Towards Explainable AI for Unsupervised Network Traffic Analysis". In: Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks. 2019.

[54] Andrea Morichetta and Marco Mellia. "Clustering and evolutionary approach for longitudinal web traffic analysis". In: *Performance Evaluation* 135 (2019), p. 102033.

[55] Andrea Morichetta and Marco Mellia. "LENTA: Longitudinal Exploration for Network Traffic Analysis". In: *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 1. IEEE. 2018, pp. 176–184.

[56] Andrea Morichetta and Marco Mellia. "LENTA: Longitudinal Exploration for Network Traffic Analysis From Passive Data". In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 814–827.

[57] Andrea Morichetta et al. "CLUE: Clustering for Mining Web URLs". In: *2016 28th International Teletraffic Congress (ITC 28)*. Vol. 01. Sept. 2016, pp. 286–294. DOI: 10.1109/ITC-28.2016.146.

[58] Antonio Murgia et al. "Lightweight Internet Traffic Classification: A Subject-Based Solution with Word Embeddings". In: Proc. of the SMARTCOMP. 2016, pp. 1–8.

[59] Thuy T. T. Nguyen and Grenville Armitage. "A survey of techniques for internet traffic classification using machine learning". In: *IEEE Communications Surveys Tutorials* 10.4 (Fourth 2008), pp. 56–76. DOI: 10.1109/SURV.2008.080406.

[60] Chiara Orsini et al. "BGPStream: A Software Framework for Live and Historical BGP Data Analysis". In: Proc. of the IMC. 2016, pp. 429–444.

[61] Mostofa Ali Patwary et al. "Scalable parallel OPTICS data clustering using graph algorithmic techniques". In: *Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. 2013, p. 49.

[62] Roberto Perdisci, Wenke Lee, and Nick Feamster. "Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces." In: *NSDI*. Vol. 10. 2010, p. 14.

[63] Lucian Popa, Ali Ghodsi, and Ion Stoica. "HTTP As the Narrow Waist of the Future Internet". In: *ACM Hotnets*. 2010.

[64] Phillip A Porras, Hassen Saïdi, and Vinod Yegneswaran. "A Foray into Conficker's Logic and Rendezvous Points." In: *LEET* 9 (2009), p. 7.

[65] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

[66] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: http://www.sciencedirect.com/science/article/pii/0377042787901257.

[67] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[68] Brett Stone-Gross et al. "Your botnet is my botnet: analysis of a botnet takeover". In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 635–647.

[69] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[70] Martino Trevisan et al. "4 Years of EU Cookie Law: Results and Lessons Learned". In: *Proceedings on Privacy Enhancing Technologies* 2019.2 (2019), pp. 126–145.

[71] Martino Trevisan et al. "Five Years at the Edge: Watching Internet from the ISP Network". In: *14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2018)*. Vol. 1. 2018.

[72] Martino Trevisan et al. "Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned". In: *IEEE Communications Magazine* 55.3 (Mar. 2017), pp. 163–169. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600756CM.

[73] Vladimir N Vapnik. "An overview of statistical learning theory". In: *IEEE transactions on neural networks* 10.5 (1999), pp. 988–999.

[74] Luca Vassio, Idilio Drago, and Marco Mellia. "Detecting user actions from HTTP traces: Toward an automatic approach". In: *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*. Sept. 2016, pp. 50–55. DOI: 10.1109/IWCMC.2016.7577032.

[75] Bingchen Wang et al. "Design and optimization of DBSCAN Algorithm based on CUDA". In: *arXiv preprint arXiv:1506.02226* (2015).

[76] Charles V Wright, Fabian Monrose, and Gerald M Masson. "On inferring application protocol behaviors in encrypted network traffic". In: *Journal of Machine Learning Research* 7.Dec (2006), pp. 2745–2769.

[77] Maarten Wullink et al. "ENTRADA: A High-Performance Network Traffic Data Stream". In: Proc. of the NOMS. 2016, pp. 913–918.

[78] Peter N Yianilos. "Data structures and algorithms for nearest neighbor search in general metric spaces". In: *SODA*. Vol. 93. 194. 1993, pp. 311–321.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was LuaLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.