Web Crawling and Processing with Limited Resources for Business Intelligence and Analytics Applications

Loredana M. Genovese, Filippo Geraci*
Institute for Informatics and Telematics, CNR, Via G. Moruzzi, 1 Pisa, Italy.

* Corresponding author. Email: filippo.geraci@iit.cnr.it
Manuscript submitted January 10, 2018; accepted March 8, 2018.
doi: 10.17706/jsw.13.5.300-316

Abstract: Business intelligence (BI) is the activity of extracting strategic information from big data. The benefits of this activity for enterprises span from the reduction of the operative costs due to a more sensible internal organization to a more productive and aware decision process. To be effective, BI relies heavily on the availability of a huge amount of (possibly high-quality) data. The steady decrease of costs for acquiring, storing and analyzing large knowledge bases has motivated big companies to invest in BI technologies. Until now, instead, SMEs (Small and Medium-sized Companies) are excluded from the benefits of BI because of their limited budget and resources. In this paper we show that a satisfactory BI activity is possible even in presence of a small budget. Our ultimate goal is not necessarily that of proposing novel solutions but providing the practitioners with a sort of hitchhiker's guide to a cost-effective web-based BI. In particular, we discuss how the Web can be used as a cheap yet reliable source of information where crawling, data cleaning and classification can be achieved using a limited amount of CPU, storage space and bandwidth.

Key words: Big data analytics, business intelligence, spam detection, web classification, web crawling.

1. Introduction

Intelligence is the activity of learning information from data. The benefits of its application to business activities have started to be discussed from the mid of the nineteen century when Hans Peter Luhn coined the term business intelligence (BI) in a paper of him [1]. In that paper Luhn, described some guidelines and functions of a business intelligence system under development at the IBM labs.

In the course of the years, BI has evolved extending outside the scope of the database community and increasing its complexity. Structured information has been replaced by unstructured one; data volumes got bigger posing new challenges both for gathering, storing and analyzing them; analytics (namely: the possibility of creating profiles from data) has emerged as a new opportunity to help companies to drive their business efforts on the most promising directions. Forecasts on future trends [2] predict an ever increasing role of BI and big data analytics in the success of companies. However, the cost of implementing BI has excluded small and medium enterprises from the benefits of big data analysis. In fact, collecting and storing data is still considered expensive and computationally demanding. Although practical guides as that in [3] provide wise suggestions to build scalable infrastructures that can optimize long-term costs, the initial effort required to setup a minimalistic BI system still remains unaffordable.

In this paper we deal with the problem of building a good quality business intelligence infrastructure keeping an eye on the company budget. Achieving this goal requires a careful choice of the data sources, extraction procedures, and pre-processing. As other studies suggest, given the universal accessibility of

contents as well as its intrinsic multidisciplinarity, the Web can be considered as the most convenient source of information. However, using the Web poses several issues in terms of data coherence, representativeness, and quality. Moreover, technical challenges have to be considered. In fact, although downloading a page seems to be for free, crawling large portions of the web may quickly become expensive and computationally demanding. Seed selection is a key factor to ensure the data to be up to date. For example, new initiatives can be promoted by registering ad-hoc websites that may not be reachable yet from other websites but are spread through other channels such as social media. Download ordering can influence representativeness while data filtering can bias coherence.

In Section 2 we discuss crawling strategies, tailored on the needs of small companies, that can drastically simplify the design of crawlers and data storage without sacrificing quality. In particular, analyzing the bottlenecks of standard crawling, we discovered that most of the CPU effort is due to the need to check whether a URL has already been discovered/downloaded or not. However, a more in-depth analysis shows that, leveraging on an ad-hoc downloading ordering, these checks can be reduced in the specific context of limited resources. This, in turn, has the effect of: simplifying the storage architecture, and providing a data stream access for the subsequent analyses.

In section 3 we try to answer to the question about whether web data is suitable for BI or not. Statistical analyses in the literature show that, although morphologically different, web corpora have the potential to be good knowledge bases once noise is filtered. We pinpoint spam and web templates as the two main sources of noise. Besides data coherence, spam has a profound impact on the consumption of bandwidth and storage resources. In order to limit resource waste, we show how to exploit DNS (domain name server) queries to predict whether a newly discovered domain is likely to be spam and, in case, remove it from the list of websites pending to be downloaded.

In section 4 we discuss the problem of per-topic classification comparing post-processing with focused crawling. In absence of resource limits the former solution would be preferable because it prevents a useful document linked by an off topic website to be lost. However, in a real scenario the higher cost of post-processing compared to its benefits makes focused crawling more feasible. In either case, however, dealing with classification requires to own a large set of examples that, at least for specialized companies, can be hard to find in publicly available ontologies or annotated resources. Thus, we show how such a set of examples can be build using semi-supervised techniques that balance accuracy and human effort.

Summarizing: in this paper we investigate the technological and methodological challenges that IT practitioners have to deal with building a BI architecture with a limited budget. We also propose solutions that can help to attain a satisfactory tradeoff between costs and BI quality.

2. Crawling

Crawling is the activity of creating a local copy of a relevant portion of the World Wide Web. This task usually represents the first challenge that IT professionals have to deal with performing BI and Web analytics. In its simplest form, a crawler is a program that receives as input a list of valid URLs (called seeds) and starts downloading and archiving the corresponding websites. Intra-domain hyperlinks are immediately followed in breath-first order, while links to external resources are appended to the list of seeds. The crawling process terminates once the list of seeds becomes empty. Despite simple in principle, the variety of purposes for which data is collected as well as technical issues, made crawling a rich and complex research field. As a measure of the interest for this topic, in a recent survey [4] the authors claim they censused 1488 articles about crawling. Restricting to the real implementations the authors listed 62 works. In the following section we dissect the main crawling design strategies and propose cost-effective options tailored on the BI process.

2.1. Seed Selection

The seed list is the initial set of URLs (more often websites' home pages) taken in input from a crawler starting to build the web image. Despite not in-depth discussed in the scientific literature about general purpose crawling, how to build a comprehensive seed list is one of the most debated topics in specialized forums.

In [5] the authors provide a successful macroscopic description of the web that can be useful to speculate on the effects of seed selection. They start recalling that the web can be modeled as a direct graph where pages are nodes and hyperlinks are edges. According to the experiments in [5], webpages can be divided into five categories that form the famous bow-tie shaped graph. The core of the web consists in the SSC: a strongly connected subgraph (i.e. a graph where given any two random nodes there exists a path connecting them). The SSC is linked from the class of IN pages and links the OUT pages. The two other relevant classes are the TENDRILS (namely: pages not connected with SSC) and the disconnected pages (i.e. pages with both in-degree and out-degree equals to 0).

Crawling the SSC and OUT components is relatively easy. In fact, given a single URL belonging to SSC it is theoretically possible to find a path that connects to all the other elements of SSC and, in turn to OUT. The elements of these categories are typically popular websites, thus finding pre-compiled lists of them on the web is a simple task.

Pages in IN and in TENDRILS, which often are new websites not yet discovered and linked, are very difficult or even impossible to be discovered. In fact, these pages have either in-degree equals to 0 or are linked from pages of the same category. According to the estimations in [5] IN and TENDRILS account for about half of the entire web. Thus, in order to crawl as many pages as possible of these two classes, their URLs should be present in the initial seed list.

Collaborative efforts have produced large lists of domains freely available on the web. Despite useful to increase the coverage of the IN and TENDRILS classes, these resources cannot be kept updated because of the dynamicity of the web. Recently, harvesting URLs posted on Twitter has emerged as an effective alternative source of fresh and updated seeds. A first application of Twitter URLs to enhance web searching is shown in [6], the ability of this social media to immediately detect new trends is proven in [7], while a machine learning approach to identify malicious URLs on Twitter posts is described in [8].

2.2. URL Crawling Order

In the context of limited resources, the URL download order matters [9]. Pioneering papers like [10] indicate the breath-first as the strategy that provides better empirical guarantees to quickly download high quality (in terms of page rank) pages. The intuition at the basis of this idea is that page quality is somehow related to the depth from the document root [11].

In [11] the authors provide a nice comparison of several ordering strategies. According to [11], the main qualifying feature to classify ordering algorithms is the amount of information they need, going from: no pre-existing information to the request of historical information about the web-graph structure.

In the context of BI for small and medium enterprises, however, supposing the existence of historical information (i.e. a previous crawling) may be inappropriate. In fact, small companies are not supposed to be able to refresh their crawling too often. Thus, even if existing, historical information could be old enough to become misleading. Hence, ordering strategies that do not require pre-existing information should be preferred.

In [12] the authors propose a hybrid approach in which websites are kept sorted using a priority queue and priorities are given by the number of URLs per host already discovered, but which download is still pending. Once a website is selected, a pre-defined number k of pages is downloaded. This strategy has

several advantages in terms of network usage. In fact: the number of DNS requests can be reduced thanks to caching, while the use of the `keep-alive` directive [13] reduces network latency. Experiments in [12] prove that this strategy still downloads high quality pages first.

Building upon [12] we propose an ordering strategy that can drastically simplify the crawler architecture in the practical case of limited resources. We exploit the important observation from [14] where the authors observe that chains of dynamic pages can lead to an infinite website even though the real information stored in the webserver is finite. According to this observation, setting a limit to the maximum number of pages per host appears sensible.

We thus keep the hosts sorted as in [12], but once a website is selected, the downloading process is done all at once. Some necessary caveats are discussed in the subsequent sections.

2.2.1. Politeness

Crawling etiquette would require not to affect webserver performances overloading it with downloading requests. According to the experiments in [12], a time interval of 15 seconds is appropriate for real life applications so as to balance politeness constraints with the timeout of the `keep-alive` directive. Moreover, using a time-series analysis of webserver access logs as in [15], we observed that a limited degree of tolerance to this limit could be acceptable in the initial phase of download. This comes from the observation of the browser behavior. In fact, since to be rendered, a page may require several supplementary files (i.e. CSS directives, images, script, etc.), in order to promptly display it, browsers have to download all these files in parallel. As a result, mimicking the browser download pattern can speedup crawling (at least of small websites) without influencing the webserver performance.

Despite the above considerations, our URL ordering policy still requires a large per-host degree of parallelism to maximize network throughput. In order to avoid breaking the etiquette rules because of the use of virtual hosts, however, some careful is required in the selection of the hosts that are downloaded in parallel. In fact, even low rates of requests can become problematic if they have to be served by the same physical machine. We thus modified the ordering policy described in [12] selecting the host with highest priority among those that not resolve to an IP already in download.

2.2.2. URLs management

According to the breath-first strategy, once a URL is found, a crawler has to check whether it is already in the list of those known and, if not, append it to the list. Since this check is URL-oriented, it has to be repeated for each outgoing link, both internal and external, in a webpage, resulting in a computationally demanding operation. In particular, distributed crawling architectures split the URL list among nodes, thus checking for a URL and updating the list might also imply extra network traffic.

Our crawling strategy introduces a drastic simplification in the URL management. Since a host is downloaded all at once, we do not need to keep track of each individual URL in the crawling, but it suffices per-host information. In fact, a newly discovered external URL, will be retrieved again once the corresponding website is downloaded unless the crawler reached the maximum number of pages for that host or the corresponding page was not connected to the home. However, except for the case of a disconnected page belonging to a small enough website (that we expect to be of marginal utility for BI applications), keeping the newly discovered URL does not change its crawling fate, thus it can be discarded.

We arrange URLs in two data structures: the intra-links table and the host queue. An intra-links table (see Fig. 1) is locally maintained by each individual crawling agent and destroyed once the download of the corresponding website is completed. This data structure is used to maintain the set of internal pages (sorted by discovery time). We implemented intra-links tables as fast hash sorted sets.

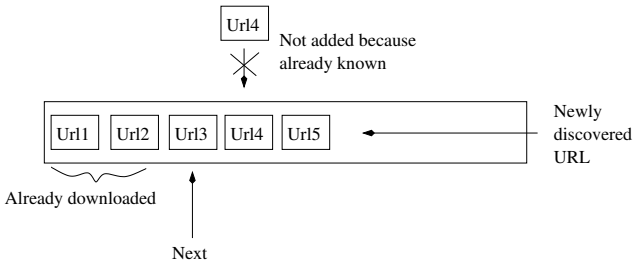

Fig. 1. Intra-link table example.

The host queue is a priority queue storing the host URL and IP address (that is looked up only when the host is selected for downloading), the status (complete, in download, pending), and the priority expressed as the number of pending downloads. In absence of the complete list of already discovered URLs, however, duplicates (namely URLs discovered more than once) cannot be detected unless accepting a degree of error, thus the number of pending downloads can only be estimated. We propose two simple estimation methods. The easier approach leverages on the idea that the importance of a website depends on the number of in-links, thus the number of pending downloads can be replaced with the host in-degree. As an alternative, a small Bloom filter can be associated to each host. Bloom filters are fast and compact data structures to sketch object sets that enable probabilistic membership query. In our case, once an external link is extracted, the Bloom filter associated to the corresponding host is queried for membership. If the URL is not found, it is added to the bloom filter and the counter of pending downloads is incremented. Despite being computationally more demanding than host in-degree, bloom filters are still much lighter than URL lists. Moreover, accessing the host queue happens only for external links that account for an average of 11.62% of the outgoing links in our experiments.

2.3. Crawling Depth

As mentioned in section 2.2, in a real scenario storage and bandwidth are limited and expensive as opposed to the possibility of dynamically create new webpages which is unlimited and for free. The authors of [14] conclude that crawling up to five levels from the home page is enough to include most of the web that is actually visited. Hypotheses on the power-low distribution of the number of pages per website [16], however, suggest that most websites are smaller. Based only upon the common sense, here we try to speculate on how the threshold on the crawling depth can be small for BI applications with severe resource limits.

Setting the depth to the minimum possible, a crawler would only download the website document root. This page, however, is often used as a sort of gateway to collect information on the user configuration and redirects the browser to the appropriate version of the home page. In this case, a crawler should be aware of redirections and follow them until reaching the proper home page.

As expected, home pages are very informative containing most of the base information about the website. In particular, leveraging only on home pages it is possible to derive some structural information about the underneath technologies [17], accomplish large scale usability tests [18], [19] and perform web classification [20].

Extending the crawling to two levels has a large impact in terms of required storage and bandwidth. In fact, home pages often tend to act as a sort of hub, thus the expected volume of pages directly reachable from the home is usually high. Based on a sample of 60 among the most popular websites, we estimated the second level of the web hierarchy to have on average 137 pages (Notice that this can be an underestimation since news websites and general-purpose portals tend to have more links than the other websites and can easily exceed 500 internal links). However, the higher cost in terms of resources is balanced by the much

richer information content. For example, ecommerce companies may have specified only commodity macro categories in the home pages leaving the subdivision by micro category or brand in the second level. Other important details like contact information and partnerships are often reported in the second level of hierarchy.

A crawling depth of three pages is often synonymous of downloading the entire website. In fact, in order to facilitate accessibility as well as indexing from the search engines, several websites use *sitemaps*. This technique consists in creating a page directly descendent from the homepage in which all the internal links of the website are enumerated. As a result, sitemaps bound the website hierarchy to three levels.

2.4. Data Storage

Data storage is probably the most complex key factor in a crawling architecture. Nonetheless, even high impact publications give few details about the storage model. For example, in [21] the authors focus on load-balancing, tolerance to failures, and other network aspects; while in [22] the authors discuss the download ordering of the pages, the updates and the content filtering. In both cases, however, storage is only mentioned. A simple classification of the huge amount of data produced during crawling is, instead, provided in [23]. Crawling data can be classified according to two main characteristics: fixed/variable size and output only/recurrent usage.

Fixed size data is easier to store and access: conventional databases can handle billions of records and serve hundreds of concurrent requests. Variable size data, instead, requires some further consideration about its usage frequency. In fact, for frequently accessed data a fast access should be preferred, while for output only data a compact representation would be better. Frequently, however, fixed and variable size data co-exist in the same record. This is the case, for example, of the record storing the information about a single webpage. In fact, according to the level of detail that the crawler provides, this record can contain several fixed size information, as for example a timestamp, and some variable size information like the page URL. However, also in this case conventional databases provide well-qualified solutions.

The most verbose variable size crawling data is the HTML content of webpages. This information, however, is used only once to extract outgoing links from the page and then it is stored for output. Small crawlers such as wget store each page (or website) in a dedicated file leaving the file system the burden of managing them. Page updates are supported by deleting the corresponding file and replacing it with a new one. Obviously, this solution cannot scale even for small crawls of few million pages (or websites).

Another simple storing strategy is that of defining a certain number of memory classes (defining buckets of fixed size) and memorizing each page in the bucket that better fits the page size. Each memory class can be stored into a separate file. This solution introduces a certain waste of space that can be minimized with a careful choice of the sizes of the memory classes. A simple method to set memory class sizes is that of downloading a small sample of pages and estimate suitable values from them. Updates are supported also in this case. However, the possibility for the new copy of a web page to change size (thus requiring a bucket of a different memory class) can introduce a form of segmentation in which certain buckets become empty. In order to avoid this phenomenon, each memory class can be endowed with a priority queue (i.e. using a min-heap) to take note of the empty slots.

As an intermediate strategy between the two previously described storage solutions, when updates are not required, HTML pages can be stored in log files. Once a new page is downloaded, it is stored just after the previous one in the log file. Keeping a record containing the length and a reference to the offset in the log file, accessing a webpage is very fast since it only requires a disk seek and a single read operation. Storage space is minimized because only two integers per page are required. Although supporting updates is possible, this model is recommended only for static crawling. In fact, updating a page would require appending the new copy to the log file and updating the corresponding record. However, the previous

version of the page would not be cancelled, thus resulting in a waste of space.

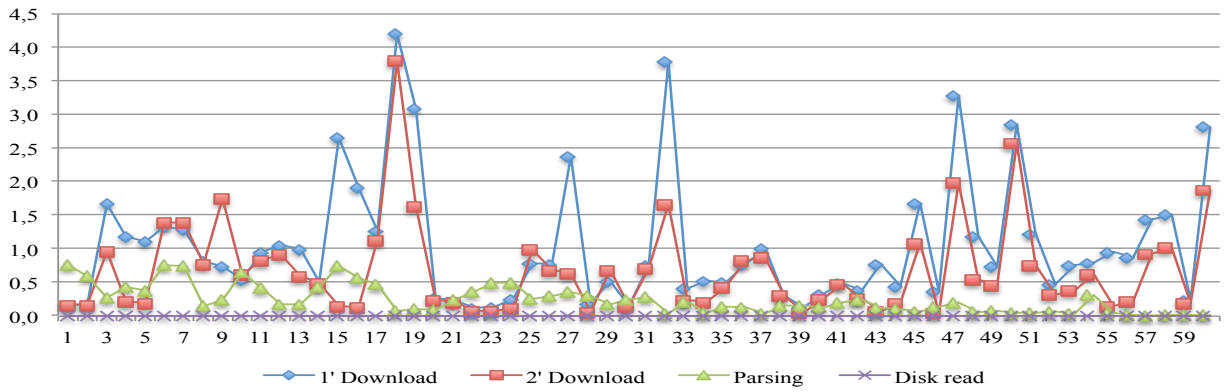

Fig. 2. Running time (in seconds) for the operations of: Download, parsing and retrieving from disk of 60 among the most popular websites.

We investigated a further class of data, not discussed in the literature, that we call auxiliary data. We define as auxiliary some piece of information that crawlers had to compute for their activity but they withdraw immediately after usage. Sometimes, however, the same information is necessary for the subsequent analysis of the crawled document corpus, thus, storing it during crawling would speed up subsequent analyses reducing the CPU requirements at the cost of a higher storage consumption. The most important information belonging to this category is parsing. As discussed later in section 2.4.1, parsing is necessary in all data analyses and can dominate the computational time. Saving it at the cost of a moderate increase of storage usage could be useful in most of the cases.

2.4.1. Parsing

As mentioned before, crawlers perform parsing to extract outgoing links from webpages. However, since this is considered a mere internal operation, general-purpose crawlers do not return parsing in output, thus forcing subsequent analysis software to redo it.

We performed a simple experiment to show how big the impact of parsing is both for crawling and data analysis. We tested the home page of 60 popular websites and compute the speed for downloading, parsing and retrieving then from disk. We repeated this operation twice so as to measure downloading speed before and after network caching. As expected, except for few exceptions (see Fig. 2) parsing is in general much faster than downloading and takes respectively 24.47% of the overall processing time for the first download and 32.64% for the second. Comparing parsing with disk access (as it would be during the analysis), instead, the situation drastically changes. In fact, in this case parsing takes on average 99% of the overall wall-clock processing time. Avoiding repeating parsing during analyses would thus enable a consistent speedup.

With the purpose of BI applications in mind, we developed an ultra-fast parser that stores the offsets of tags in a vector of quadruples (see Fig. 3). Since the tree structure of HTML is rarely necessary for analysis we do not keep track of it. However, we notice that building it would cost $O(n \log_2 n)$ where n is the number of tags in the page (in our experiment $n \cong 2988$). Each tag is described as its starting position and length (namely: the distance in bytes between the opened and the closed angle brackets). In addition, the offset of the tag name and its length is stored. In order to minimize the number of disk read operations needed to load the whole parsing data structure, we maintain in the first word of the vector the size of a quadruple and the number of tags. As a result, loading the data structure requires only two disk accesses.

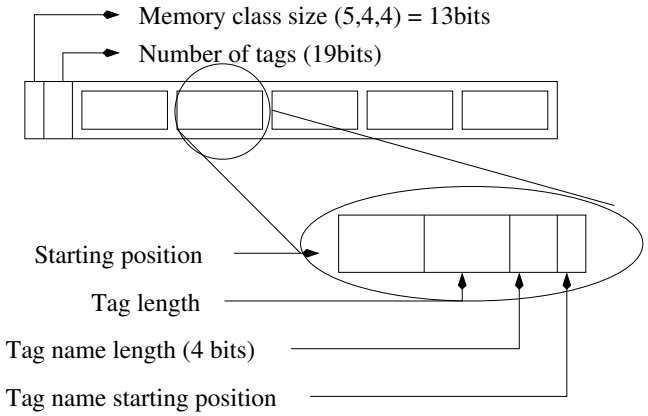

Fig. 3. Graphical description of the parsing data structure.

A relevant problem of this approach (and parsing in general) is that we cannot do any assumption on the dimensions of pages and tags. In general, except for the length of the tag name, which is limited to 10 bytes in HTML5, we should use up to 4 bytes to store each element of the quadruple. Using relative distances does not mitigate the problem. This would mean that storing the information of each tag would require 13 bytes. Given that our experiments indicate the median of the length of HTML tags to be 8 bytes and 56.27% of the tags to be smaller than 13 bytes, this would be an unacceptable blowup of required space. To deal with this problem, we use an adaptive strategy in which, after processing a page, the parser computes the minimum number of bits necessary for each field of the quadruple (except for the tag length which is kept constant to 4 bits). These values are stored in the header word of the parsing structure. In particular, we use 5 bits for the page offset (i.e. limiting the maximum page size to $2^{2^5} = ^{32} \cong 4G$ bytes), 4 bits for the tag length (i.e. limiting the tag length to $2^{16} = 65536$ bytes), 4 bits for the distance from the beginning of tag. In order to fit the header in a 32bit word, we reserve for the counter of the number of tags the remaining 19 bits. This, however, is not a limitation in practice since it would mean that our data structure can deal with pages to up to 524288 tags.

We experimentally evaluated the worst-case number of bits required to store each field. We observed that 20 bits are enough for the maximum page length, 11 for the tag length and 7 for the tag name distance. Overall a quadruple would require a total of 42 bits in the worst case.

3. Filtering

What makes skeptical BI practitioners in using the Web as a source of information for their analyses is the untrustworthiness of sources and possibly the low text quality due to the presence of every sort of artificially added content in the data [24] (aka boilerplate). Artificial text is often used to try to circumvent the ranking algorithms of search engines so as to obtain a privileged position in the list of results. Other sources of unrelated content are: interfaces (menus, navigation bars, etc.) advertising banners and parked domains.

In order to discern whether it is possible to use web data for BI or not, we thus have to answer to some questions:

1) Are web-based corpora qualitatively comparable with traditional ones?
2) Can web-based corpora influence the BI activity?

In [25], the author tries to answer to the first question carrying out a comparison of “traditional” corpora and web-based ones, focusing on Czech and Slovak. Experiments show that none of the compared corpus is morphosyntactically superior. Web-based corpora are simply different. A further insight comes from [26]

where the author shows the effect of the size of corpus. The most interesting result is the empirical demonstration that when the dataset is large enough, the language of the web perfectly reflects the standard written language. Lowering the size of the dataset, the equality persists for the most frequent nouns while differences begin to appear in the least frequent nouns.

A partial answer to the second question is provided in [27] where the authors prove that a computer-mediated text can deceive the reader. However, this category of text has specific linguistic features that can be easily detected and removed [28].

We observe that most of the uninteresting text in web-based corpora comes from two main sources: templates (which include advertisement banners, navigational elements, etc.) and spam websites (aka parked domains). In [29] we proposed a general-purpose algorithm for template removal. A brief discussion of the benefits for BI of our approach as well as a sketch of the algorithm is presented in section 3.1. Differently from approaches like that in [30] our approach is not technology-specific and, thus, it is not supposed to become deprecated with the evolution of web technologies. Moreover, integrating our approach with our crawling ordering described in section 2.2, it is possible to reduce the storage requirements of crawling still enabling to restore the original webpages.

In [31] we proposed a clustering-based approach to detect the most common category of spam: that of DNS-based parked domains. Our experiments showed that these websites resolve to dedicated name servers. As a result, filtering at DNS level would introduce a marginal loss of useful information while being much more efficient than a per-website classification. However, narrowing to only this category of spam is not enough for BI applications. In section 3.2.2 we extend the approach in [31] to all the categories of spam websites.

3.1. Template Removal

Since the advent of content management systems (CMS) in the late nineties, web sites started using templates. Estimations in [32] rate 48% of the top websites as using CMSs, while the number of sites that employ templates is probably much higher. Web templates consist in stretches of invariable HTML code interlaced with the main text of the page. The goal of this technology is that of providing a uniform look-and-feel to all the pages of a site easing the website manager from the heavy task of manually curating it. Thus, templates do not introduce useful information from the BI point of view, but have a mere role of improving the user experience. However, the impact of templates in the amount of resources used to store websites is not negligible. According to [29] the wasted space due to templates is highly variable and can span from 22% (estimated on en.wikipedia.org) to 97% (estimated on zdnet.com).

A further undesired effect of web templates is that they have been reported to negatively affect the accuracy of data mining tools because of the noise they introduce in the text [33]. In fact, large templates can introduce a not marginal number of extra words in the webpage. These words, in turn, can either mystify duplicate detection algorithms or artificially bias the similarity of documents.

Consequently, stripping templates from webpages is expected to produce a drastic saving of storage consumption as well as an increase of the data quality for the subsequent BI process.

3.1.1. An algorithm for template removal

In this section we sketch out the main ideas behind our template extraction algorithm leaving details to the original work in [29]. Our algorithm assumes that the template of a website is made up of a set of overrepresented stretches of HTML tokens (namely: a tag or a piece of text between two tags) and consists of two main routines: the *aligner* and a bottom-up *template distillation algorithm*. The aligner takes in input two tokenized webpages and uses the result of the Needleman and Wunsch global sequence alignment algorithm [34] to build a *consensus page* containing the union of the tags of the two pages. Each tag is

endowed with a numerical value representing the frequency of the tag in the alignment. Starting from a set $P = (p_1,\ldots,p_t)$ of t pages, the *template distillation algorithm* (see figure 4) uses a bottom up approach that recursively halves the number of elements in P aligning pairs of pages. Once P contains only one page, the tags with estimated frequency higher than 0.5 (i.e. the tags that are present in at least half of the original t pages) are returned as the website template. For the ease of computation, t is constrained to be a power of 2.

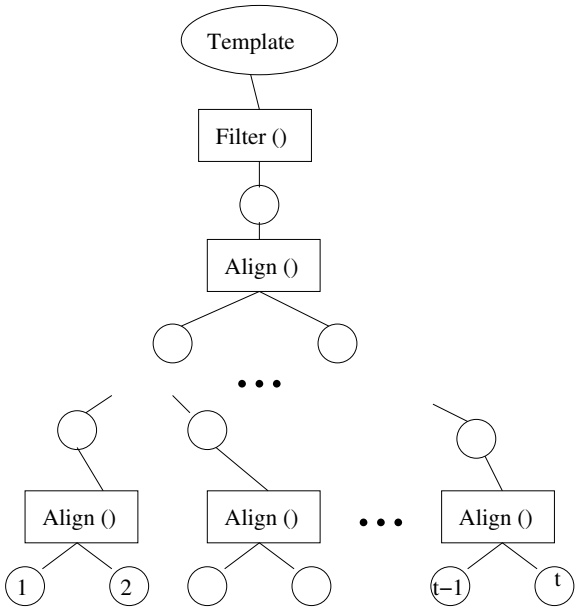

Fig. 4. Example run of the template distillation algorithm.

3.2. Spam Websites Removal

As mentioned in Section 3, spamming is one of the two major sources of noise in a web-based corpus of documents for BI applications. Although spam websites no longer look like long flat lists of ads, every Internet user gathered experience on how to recognize them at a glance. In fact, most spam websites tend to look similar to each other. Exploiting this fact, in [31] we developed a clustering-based approach to identify them.

Going further, we observed that it is possible to leverage on the mechanism used to create spam websites on a large scale to predict whether a website would contain spam or not without downloading it. In fact, big providers of parking facilities only require an ad-hoc setting of the website name server (NS) to supply the service. As shown in section 3.2.3, classifying domains according with their NSs leads to an accurate and computationally cheap method to remove spam saving bandwidth and storage.

3.2.1. A clustering algorithm for spam detection

In this section we sketch our approach leaving details to the original work in [31]. According to the intuition that spam websites tend to look similar (not equal) to each other, given a model page and notion of distance among pages, all spam websites should form a single homogeneous cluster.

However, the reality is a bit more complicated. In fact, parking service companies provide several templates that the customer can use and personalize. Consequently, spam websites do not partition into a single cluster, but form a large number (as large as the number of templates and customizations) of homogeneous and well-separated clusters. Distilling and keeping updated a comprehensive list of templates is unfeasible, thus a model-based approach is not pursuable. Yet, to save resources a website should be classified before downloading it, thus the classification mechanism should be based on features that can be

known in advance.

Based upon the above considerations, our algorithm takes in input a set of home pages resolved from the same name server and classify it as a regular ISP or as a DPP (Domain Parking Provider). The list of DPPs can subsequently be used during crawling to filter out domains resolving to these NSs since they are likely to be spam.

As a clustering algorithm we used our own modified version of the further-point-first (FPF) algorithm for the k-center problem (namely: find a subset of k input elements to use as cluster centers so that to minimize the maximum cluster radius) originally described in [36]. Our implementation returns the same result of the original version, however, it drastically reduces the number of distance computations leveraging on the triangular inequality in metric spaces. Thus, to use our algorithm it is necessary that the distance function defines a metric space.

Let P be an input set of pages and let $C = \{c_1,...c_k) \subset P$ be the (initially empty) subset of cluster centers. The first center is chosen at random among the elements in P. Then, as a new center it is selected the element $p \in P$ such that the distance $\min_j d(p,c_j)$ is maximized. The procedure stops when the desired number of centers is selected, then the remaining elements of P are assigned to the closest center.

In its simplest form, the distance function $d()$ should measure the proportion of tags in common between two pages given their overall length. Keeping track of the relative ordering of the tags in the pages requires using the quadratic algorithm described in [37]. However, $d()$ can be estimated with a computationally less expensive function $f()$ that (ignoring the relative ordering of the tags) computes the Jaccard score of the frequency vector (a vector that for each tag counts the number of occurrences) of the pages. Since it holds that $f(x,y) \leq d(x,y)$, if the value of $f()$ is high enough to decide that a certain page is too far from a center, the computation of $d()$ becomes unnecessary.

3.2.2. Detecting non-NS based spam websites

The method described in [31] focuses only on spam websites parked through DNS-level redirection. Narrowing only to crawling purposes, this choice is agreeable because classification can be done before the download, thus preventing wasting space and bandwidth. For BI applications, however, all types of redirections must be taken into account. Besides DNS-based, there are two other possible redirection types (see [35] for details): at HTTP level and at HTML level.

HTTP redirections are directly managed by web servers that return a 3XX class status code and a target URL while HTML redirections are small pieces of code aimed at causing the browser not to generate output, but immediately load another web page following the appropriate URL (i.e. using the tag *meta* setting refresh at zero time, calling a JavaScript function updating either the *href* or the *location* variable, or introducing a full-window *iframe* tag). Independently from the type, both redirection types cause the browser to land to a target URL belonging to a DPP website [38].

Spam websites using non-DNS redirections can still be detected and filtered out through some considerations on the distribution of in-links per host. According to the famous model of websites in [39] there exist two important categories of websites: hubs (websites with several out-links) and authorities (websites with several in-links). Let $C(\gamma)$ be the set of pages belonging an authority website γ. We can suppose that γ is a spam provider if it satisfies the following conditions:

1) Most of the in-links of γ are due to redirections
2) The pages in $C(\gamma)$ cluster evenly.

Condition 1 prevents from considering spam those pages that link to popular services, while condition 2

is the same hypothesis at the base of our approach.

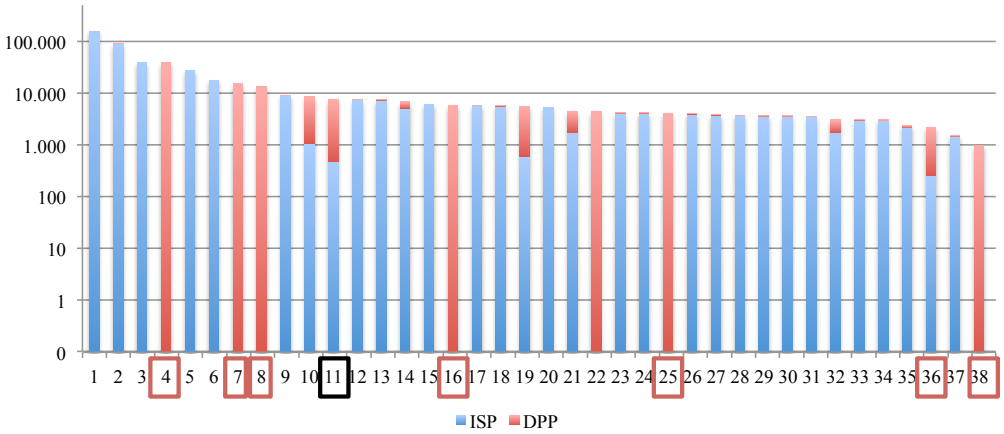

Fig. 5. Number of hosts per Name Server (NS). The blue bar indicates regular websites, the red bar indicates parked domains. The red boxes on the x axe represent DPP NSs, the black box a misclassified NS.

3.2.3. Evaluation

We performed an experiment to quantify: the fraction of relevant pages that would be lost using our approach and the volume of irrelevant pages that would be wrongly left in the corpus. We analyzed a crawling of about 2 million valid web-pages hosted by more than 100 thousand different primary name servers (NS) and distributed among the four TLDs ".it", ".com", ".org" and ".net". We than focused on the most represented NS, which are, indeed, those that more likely could host spam websites, removing those with less than 1000 pages. We obtained a corpus of 541,138 valid home pages distributed among 38 name servers. Subsequently, we used the computer-aided approach described in [31] to classify the home pages in two categories: irrelevant/relevant. As figure 5 confirms, most NSs (we obfuscated the full names for privacy reasons) focus only on one business, either regular hosting or domain parking, with only four notable exceptions: NS10, NS11, NS19 and NS36. NS10 and NS19 are two medium-size private NSs belonging to two Italian TV production companies that registered a large number (respectively 87.15% and 89.19%) of dictionary-based domain names for own future use. NS11 and NS36, instead, are specialized companies for domain parking that also hosts a fraction (6.23% and 11.43%) of own regular pages. Notice that NS11 is the only NS misclassified using our method.

Table 1. Per-Website Confusion Matrix

		Human Classification		Total
		Relevant	Irrelevant	
Algorithm	Relevant	430,676	25,464	456,140
	Irrelevant	253	84,745	84,998
Total		430,929	110,209	541,138

In Table 1 we show in detail the benefits of using our method to filter out spam websites. Out of the 25,464 irrelevant pages misclassified, 7408 comes from NS10 and 4897 comes from NS19. Although these pages are indeed not interesting for the human judge, they cannot properly be considered as spam since they are not simple lists of ads but still contain human-generated text. Only 7118 pages can directly be ascribed to software misclassification.

The fraction of relevant content that would be lost is negligible and consists only in 253 pages. This result is particularly important for the class BI applications in which the discovery of not yet emerging new phenomena is crucial.

As a further important observation, since a NS-based classification does not require downloading and storing webpages, in the case of our experiment the elements of the NSs classified as DPP would not be downloaded, thus saving 15.7% of crawling resources.

Summarizing: experiments show that using the NS to classify whether the corresponding websites are regular or spam provides a reasonable accuracy (as high as 95.25%) as well as it is computationally convenient.

4. Classification

Letting a general-purpose crawler loose, it will start downloading the web without discriminating among topics until stopped or until its storage/bandwidth resources are completely exhausted. The resulting corpus will consist in a huge mass of documents spreading on almost all the human knowledge. However, since typical BI applications need to focus only on documents about the investigated topic, a mechanism to filter out unnecessary websites is necessary.

Filtering can be applied either at crawling time (focused crawling) or ex post. In the first case, once a website is discovered, the crawler classifies it and decides whether to download it or not. The most important advantage of this strategy is that of saving resources. In fact, if a website is classified as off topic, it is neither crawled nor stored. On the other hand, discarding a website could cause links to interesting pages to be irreparably lost. A posteriori filtering is much more resource demanding than focused crawling and should be preferred only in those situations in which the application requires to crawl a sample as complete as possible. Independently from the approach, however, a mechanism to classify websites is necessary.

4.1. Creation of the Training Set

Supervised classification is the task of assigning a document to a class based on pre-existing examples. This tool has successfully been used in focused crawling as well as in general website classification. The hunger of classifiers for training examples, however, poses severe limitations to the practical usage of supervised classification. In fact, building a large-enough high quality training set can require a long and tedious manual classification by a domain expert. In order to reduce human involvement in the labeling of examples, some authors proposed approaches that leverage on existing knowledge bases. In [40] and [41] the authors use ontologies to drive crawling. In particular, in [40] the authors show how to exploit the newly discovered documents to extend the ontology obtaining an increasingly more accurate training set; while in [41] the authors describe a framework to compute the relevance of a document with an ontology entity. In [42] the authors experimentally prove that website classification can be done without scarifying accuracy but only using positive examples. All those works, however, have in common the assumption that such form of domain knowledge exists and is available.

The above assumption is acceptable dealing with general topics that are more likely to be covered by such a knowledge base. Small and medium enterprises, instead, may be interested in uncommon specialized topics not enough represented in the existing freely available datasets. Using clustering in place of a supervised approach would mean trading the classification accuracy with the simplicity and ubiquity of the methodology.

In [20] we faced the problem of using a small description of a class (as small as a weighted list of keywords) to automatically derive from the Web a list of pages that are more likely to be good examples to train a classifier. As most semi-supervised methods, our approach attempts to balance the use of human resources with the decrease of classification accuracy. In fact, to the easy task of compiling a list of keywords corresponds the introduction of some misclassified examples and, in turn, a slight reduction of accuracy.

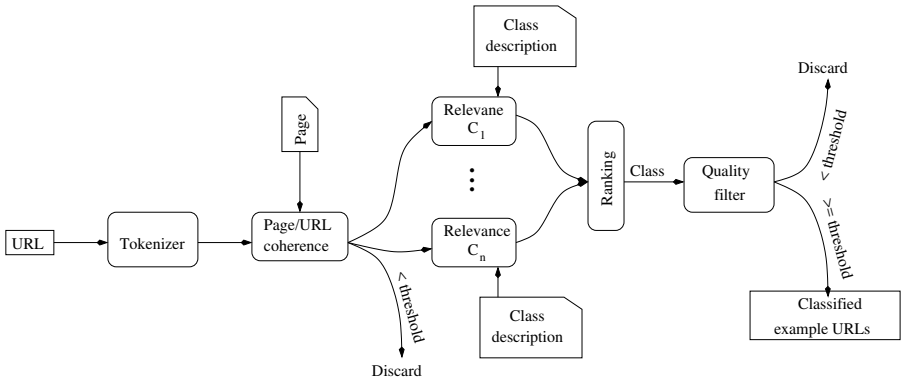

Fig. 6. Graphical representation of the labeling pipeline.

4.1.1. A semi supervised pipeline for the extraction of training examples

In this section we sketch out our pipeline leaving details to the original work in [20]. Both the plethora of informal guides on how to choose a domain name and the scientific literature on the psychological effects of brand names agree that a successful brand should consist in the juxtaposition of few words. In particular, in [43] the authors perform an experiment in which they discuss the effects of using meaningful brand names. Results show how names conveying the benefits of the product (i.e. smart phone) achieve a commercial advantage on other brands. Going a step further, in [44] the authors analyze the linguistic characteristics of brand names concluding that semantic appositeness is one of the three main features that contribute to make the brand familiar and, in turn, successful. Although popular brands of the net economy coined in the mid-nineties represent eminent exceptions to the above studies, it is realistic to hypothesize the abundance of domain names consisting in meaningful short sentences semantically related to the website content.

Given a set $C = \{c_1, \ldots c_n\}$ of n classess, each of which described as a document, and a URL tokenized as a list of keywords, our approach maps the problem of assigning a website to a class as the problem of ranking classes according to their relevance to the URL text. Figure 6 graphically depicts the sequence of steps of our pipeline.

We used a dictionary-based greedy algorithm to distill a list of words from a URL. The domain name is scanned from left to right until a dictionary word is found, then the word is extracted and the procedure is recoursively invoked on the ramaining part of the URL. In order to avoid misleading or trivial tokenizations (i.e. “an acid” instead of “anacid”), our algorithm searches words in length decreasing order. If the procedure does not succeed to use all the characters of the domain it backtracks to test different alternatives. If no complete tokenization is found, the URL is discarded and the corresponding website is no longer considered as a potential candidate to be added to the training set.

URL labeling is done using a learning to rank approach [45]. This framework provides a background to solve the problem of sorting a set of documents according to a pool of measures of relevance to a query. Each relevance measure induces a ranking that is averaged with the others in order produce a final ranking. In our case, we used the class descriptions in place of the set of documents and the text extracted from the URL as the query. Relecance is computed through: cosine similarity, generalized Jaccard coefficient [46], dice coefficient, overlap coefficient, Weighted Coordination Match Similarity [47], BM25 [48], and LMIR_{DIR} [49]. As label we chose the class with highest ranking, while as a degree of membership we used the average of the relevance measures normalized to the range [0,1].

In order to minimize the number of mislabelled examples, and in turn bias the subsequent learning, the labelling pipeline implements two filters: after tokenization and after ranking. The first filter consists in measuring the degree of page/URL coherence: our algorithm computes the cosine similarity between the text derived from the URL and the page content, then it verifies that this measure exceeds a user-defined

threshold. The second filter ensures that the class assignment is neither weak nor ambiguous. In particular, weakness is measured as the inverse degree of membership with the labelling class, while ambiguity is evaluated as the number of classes such that the degree of membership exceeds a pre-defined threshold.

Experiments in [20] over 20,016 webpages belonging to 9 non-overlapping categories show that 32.36% of the URLs succeed to be tokenized and 26.33% of the pages are labeled with an accuracy of 85.8%.

5. Conclusion

In this paper we addressed the problem of enabling BI and big data analytics also for those small and medium enterprises that, because of their limited budget and IT resources, until now have been excluded from the benefits of these activities. In particular, we observed that, despite morphologically different from standard text corpora, using the Web as a source of information does not necessarily lead to low quality data. In fact, applying convenient template removal filters as well as spam filters, it is possible to remove undesired text/pages without altering the useful informative content of the Web. We also showed how careful design choices can contribute to drastically reduce the cost of crawling, and in turn, allow to download a large-enough portion of the Web so as to make BI statistically significant. Finally, we dealt with the problem of per-topic website classification showing that a good tradeoff between the human effort of creating a manual description of the classes and the classification accuracy is possible. Summarizing: our experience shows that accurate resource-driven design choices can lead to a satisfactory BI activity also for small and medium businesses without sacrificing quality.

Acknowledgment

This work was supported by: the Regione Toscana of Italy under the grant POR CRO 2007/2013 Asse IV Capitale Umano and the Italian Registry of the ccTLD .it" Registro.it.

References

[1] Luhn, H. P. (1958). A business intelligence system. *IBM J. Res. Dev*, 314-319.

[2] Hsinchun, C., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS quarterly 36.

[3] Hu, H., Wen, Y, Chua, T., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*. Vol. 2, 652-687.

[4] Kumar, M., Bhatia, R., & Rattan D. (2017). A survey of Web crawlers for information retrieval. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

[5] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., & Wiener, J. (2000). Graph structure in the web. *Computer Networks*, *33(1)*, 309-320.

[6] Rowlands, T., Hawking, D., & Sankaranarayana. R. (2010). New-web search with microblog annotations. *Proceedings of the 19th International Conference on World Wide Web*.

[7] Aiello, L. M., Petkos, G., Martin, C., Corney, D., Papadopoulos, S., Skraba, R., Oker, A., Kompatsiaris, I., & Jaimes, A. (2013). Sensing trending topics in Twitter. *IEEE Trans. on Multimedia*.

[8] Wang, D., Navathe, S. B., Liu, L., Irani, D., Tamersoy, A., & Pu, C. (2013). Click traffic analysis of short url spam on twitter. *Proceedings of the 9th Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)*.

[9] Cho, J., Garcia-Molina H., & Page, L. (1998). Efficient crawling through URL ordering. C*omputer Networks and ISDN Systems*.

[10] Najork, M., & Wiener, J. L. Breadth-first crawling yields high-quality pages. *Proceedings of the 10th international conference on World Wide Web (WWW '01)*.

[11] Baeza-Yates, R., & Castillo, C., Marin, M., & Rodriguez, A. Crawling a country: Better strategies than breadth-first for web page ordering. *Proceedings of the Special interest tracks and posters of the 14th Int. Conf. on World Wide Web (WWW '05)*.

[12] Castillo, C., Marin, M., Rodrıguez, A., & Baeza-Yates, R. (2004). Scheduling algorithms for Web crawling.

[13] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). RFC 2616 - HTTP/1.1, the hypertext transfer protocol. http://w3.org/Protocols/rfc2616/rfc2616.html

[14] Baeza-Yates, R., & Castillo, C. (2004). Crawling the infinite web: Five levels are enough. *Algorithms and Models for the Web-Graph*.

[15] Iyengar, A. K., Squillante, M. S., & Zhang, L. (1999). Analysis and characterization of large — Scale Web server access patterns and performance.

[16] Adamic L. A., & Huberman, B. A. (2001). The Web's hidden order. *Commun*.

[17] Gomes, D., Nogueira, A., Miranda, J., & Costa, M. (2009). Introducing the Portuguese web archive initiative. *In 8th International Web Archiving Workshop*.

[18] William, A., & Tullis, T. (2013). Measuring the user experience: collecting, analyzing, and presenting usability metrics. *Newnes*.

[19] Lopes, R., Gomes, D., & Carriço, L. (2010). Web not for all: A large scale study of web accessibility. *Proceedings of the Int. Cross Disciplinary Conference on Web Accessibility*.

[20] Geraci, F., & Papini, T. (2017). Approximating multi-class text classification via automatic generation of training examples. *Proceedings of the 18th International Conference on Computational Linguistics and Intelligent Text Processing*.

[21] Boldi, P., Codenotti, B., Santini, M., & Vigna, P. (2004). Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience*.

[22] Olston, C., & Najork, M. (2010). Web crawling. *Foundations and Trends® in Information Retrieval 4.3*.

[23] Felicioli, C.., Geraci, F.., & Pellegrini, M. (2011). Medium sized crawling made fast and easy through Lumbricus webis. *Int. Conf. on Machine Learning and Cybernetics*.

[24] Gyongyi, Z., & Garcia-Molina, H. (2005). Web spam taxonomy. *1st Int. Workshop on Adversarial Information Retrieval on the Web AIRWeb*.

[25] Benko, V. (2017). Are web corpora inferior? The case of Czech and Slovak. *Proceedings of the Workshop on Challenges in the Management of Large Corpora and Big Data and Natural Language Processing*.

[26] Khokhlova, M. (2016). Large corpora and frequency nouns. *Proceedings of the Int. Conf. on Computational Linguistics and Intellectual Technologies: "Dialogue 2016*.

[27] Zhou, L., & Burgoon, J. K., Nunamaker, J. F., & Twitchell, D. (2004). Automating linguistics-based cues for detecting deception in text-based asynchronous computer-mediated communications. *Group decision and Negotiation*.

[28] Piskorski, J., Sydow, M., & Weiss, D. (2008). Exploring linguistic features for web spam detection: A preliminary study. *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '08)*.

[29] Geraci, F., & Maggini, M. (2011). A fast method for web template extraction via a multi-sequence alignment approach. *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*.

[30] Schafer, R. (2017). Accurate and efficient general-purpose boilerplate detection for crawled web corpora. *Language Resources and Evaluation*, *51(3)*, 873-889.

[31] Geraci, F. (2015). Identification of web spam through clustering of website structures. P*roceedings of the 24th International Conference on World Wide Web*.

[32] W³Techs, Usage of content management systems for websites.

https://w3techs.com/technologies/overview/content_management/all/
[33] Martin, L., & Gottron, T. (2012). Readability and the Web. *Future Internet 4.1*.
[34] Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*.
[35] Almishari, M., & Yang, X. (2010). Ads-portal domains: Identification and measurements. *ACM Trans. Web*, *4(2)*.
[36] Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *In Theoretical Computer Science*.
[37] Myers, E. W. (1986). An O (ND) difference algorithm and its variations. *Algorithmica* 1.1 (1986).
[38] Li, Z., Alrwais, S., Xie, Y., Yu, F., & Wang, X. (2013). Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. *IEEE Symposium on Security and Privacy*.
[39] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of ACM*.
[40] Luong, H. P., Gauch, S., & Wang, Q. (2009). Ontology-based focused crawling. *Proceedings of the Int. Conf. on Information, Process, and Knowledge Management*.
[41] Ehrig, M., & Maedche, A. (2003). Ontology-focused crawling of Web documents. *Proceedings of the 2003 ACM symposium on Applied computing (SAC '03)*.
[42] Yu, H., Han, J., & Chang, K. C. (2004). Pebl: Web page classification without negative examples. *IEEE Transactions on Knowledge and Data Engineering*, *16(1)*.
[43] Keller, K. L., Heckler, S. E., & Houston, M. J. (1998). The effects of brand name suggestiveness on advertising recall. *The Journal of Marketing*.
[44] Lowrey, T. M., Shrum, L. J., & Dubitsky, T. M. (2003). The relation between brand-name linguistic characteristics and brand-name memory. *Journal of Advertising*.
[45] Hang, L. (2011). A short introduction to learning to rank. *IEICE TRANS. on Information and Systems*.
[46] Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*.
[47] Wilkinson, R., Zobel, J., & Sacks-Davis, R. (1995). Similarity measures for short queries.
[48] Robertson, S. E. (1997). Overview of the okapi projects. *Journal of Documentation*.
[49] Bennett, G., Scholer, F., & Uitdenbogerd, A. (2008). A comparative study of probabilistic and language models for information retrieval. *Proceedings of the 19th Cconf. on Australasian Database*.

Loredana M. Genovese was born in Crotone, Italy, in June 1971. She received her master degree in Computer Science from the University of Pisa in 2002 with a thesis on parallel replicated databases. She was also awarded a ``laurea specialistica'' degree in Informatics in 2004. After several years spent on a private internet company, she joined the Institute for Informatics and Telematics of the Italian National Research Council in 2013 with the position of research assistant.

Filippo Geraci was born in Sicily, Italy, in September 1977. He received his degree in computer science from the University of Pisa and his Ph.D. in information engineering from the University of Siena. He is currently researcher at the Institute for Informatics and Telematics of the Italian National Research Council in Pisa. He held the chair of the course of Information systems for business management at the University of Siena. He is tutor of the course of Algorithm Design at the University of Pisa. His research fields are: algorithms for the web, and bioinformatics.