CALA: ClAssifying Links Automatically based on their URL

Inma Hernández [a, *], Carlos R. Rivero [b], David Ruiz [a], Rafael Corchuelo [a]

[a] University of Sevilla, ETSI Informática, Avda. Reina Mercedes s/n, Sevilla E-41012, Spain
[b] Rochester Institute of Technology, Department of Computer Science, 102 Lomb Memorial Dr., Rochester, NY 14623-5608, USA

A B S T R A C T

Web page classification refers to the problem of automatically assigning a web page to one or more classes after analysing its features. Automated web page classifiers have many applications, and many researchers have proposed techniques and tools to perform web page classification. Unfortunately, the existing tools have a number of drawbacks that makes them unappealing for real-world scenarios, namely: they require a previous extensive crawling, they are supervised, they need to download a page before classifying it, or they are site-, language-, or domain-dependent. In this article, we propose CALA, a tool for URL-based web page classification. The strongest features of our tool are that it does not require a previous extensive crawling to achieve good classification results, it is unsupervised, it is based exclusively on URL features, which means that pages can be classified without downloading them, and it is site-, language-, and domain-independent, which makes it generally applicable. We have validated our tool with 22 real-world web sites from multiple domains and languages, and our conclusion is that CALA is very effective and efficient in practice.

Keywords:
Web page classification
URL patterns

1. Introduction

Web page classification refers to the problem of automatically assigning a web page to one or more classes after analysing its features. Automated web page classifiers have many applications, either for human decision support or to be integrated into another automated processes. The most usual are the following: (1) endowing Virtual Integration crawlers with the intelligence to determine whether a web page may contain or not information that is relevant to a query (Blanco et al., 2011; Li and Zhong, 2004; Vidal et al., 2008; Hernández et al., 2014), (2) applying the most appropriate extraction model to retrieve information from a certain web page (Crescenzi et al., 2001), (3) filtering web pages to avoid certain types of contents, specially advertisements (Shih and Karger, 2004), (4) devising parental control systems (Zhang et al., 2006), detecting and canonicalising duplicated URLs (Bar-Yossef et al., 2009; Koppula et al., 2010), (5) constructing, maintaining or expanding web directories, e.g., dmoz.org or Yahoo! Directories (Dumais and Chen, 2000; Shen et al., 2004), or (6) devising focused crawlers that retrieve web pages on a certain topic in an efficient way (Chakrabarti et al., 1999; Xu et al., 2014).

Web page classifiers are usually learnt from a training set, which is a dataset of selected web pages. Depending on whether the pages in the training set have a pre-defined class or not, the techniques to learn classifiers are catalogued as supervised or unsupervised. Note that it is very common to use the expression "a supervised classifier" or "an unsupervised classifier" to refer to classifiers that were learnt using supervised or unsupervised techniques, respectively. In this article we use these expressions since they are so common that they cannot induce any confusion. Supervised classifiers require the user to annotate the training set, i.e., they require the user to analyse every page in the training set and assign it to one or more classes; this is usually considered one of the main problems with supervised classifiers, since annotating the training set is usually tedious, time-consuming, and error-prone. Contrarily, unsupervised classifiers work on a training set in which the web pages have not been pre-classified by the users (Jain and Dubes, 1988). This problem is far more difficult to solve since there is no information about the classes. These techniques are based on the concept of distance between the elements to be classified (Deza and Deza, 2012); in general, they try to find a set of classes such that the pages that belong to a class are as close as possible to each other, but as distant as possible from the pages in the other classes (Xu and Wunsch, 2005). Unsupervised classifiers are appealing insofar they relieve the users from the burden of annotating the training set, but require them to analyse the resulting classes and assigning a meaning to them (which hopefully requires much less effort than annotating a training set).

* Corresponding author. +34 954552770; fax: +34 954557139.
E-mail addresses: inmahernandez@us.es (I. Hernández), crr@cs.rit.edu (C.R. Rivero), druiz@us.es (D. Ruiz), corchu@us.es (R. Corchuelo).

The literature provides a variety of tools to classify web pages. Regarding the types of features used, they can be broadly classified into the following categories: term-based tools (Beil et al., 2002; Hotho et al., 2002; Kwon and Lee, 2003; Selamat and Omatu, 2004), structure-based tools (Bar-Yossef and Rajagopalan, 2002; Blanco et al., 2008; de Castro Reis et al., 2004; Vieira et al., 2006), visual-based tools (Zhu et al., 2008; Fersini et al., 2008; Kovacevic et al., 2002), link-based tools (Bhagat et al., 2007; de Campos et al., 2008; Getoor et al., 2001; Zhu et al., 2007; Xie et al., 2007), and URL-based tools (Brin, 1998; Kan and Thi, 2005; Baykan et al., 2011; Shih and Karger, 2004; Koppula et al., 2010; Vidal et al., 2008; Bar-Yossef et al., 2009; Blanco et al., 2011; Gollapalli et al., 2015; Kenekayoro et al., 2014; Abdallah and de la Iglesia, 2014). The tools in the first three categories rely on content-based features, i.e., they require to download a web page prior to analysing its terms, structure, or visual features; this makes them of little interest for real-world web sites, since downloading web pages before classifying them puts a load on the server, and consumes bandwidth, even if the page turns out to be irrelevant and needs to be discarded. Link-based tools build on analysing the graph of links amongst the pages of a web site; thus, they require to perform an extensive crawling in order to learn a classifier. This makes the previous tools unappealing to deal with real-world web sites that usually comprise a significant number of web pages, which has motivated many authors to focus on URL-based tools.

Classifying a web page building on features of its URL is appealing insofar it can be classified without actually downloading it, which has a positive impact on performance (Baykan et al., 2009). Unfortunately, after analysing the existing tools in the literature, we have found that they have a number of drawbacks: almost every tool that we have analysed requires a previous extensive crawling of the web site (Brin, 1998; Kan and Thi, 2005; Baykan et al., 2011; Vidal et al., 2008; Bar-Yossef et al., 2009; Blanco et al., 2011; Gollapalli et al., 2015; Kenekayoro et al., 2014), most of them are supervised (Brin, 1998; Kan and Thi, 2005; Baykan et al., 2011; Vidal et al., 2008; Bar-Yossef et al., 2009; Abdallah and de la Iglesia, 2014; Kenekayoro et al., 2014), while others need to download the page to compute content-based features that help them achieve good performance (Blanco et al., 2011; Shih and Karger, 2004; Gollapalli et al., 2015; Kenekayoro et al., 2014). The previous issues have a negative impact on scalability. Furthermore, many of these tools are either site-, language-, or domain-dependent (Baykan et al., 2011; Kan and Thi, 2005; Shih and Karger, 2004; Abdallah and de la Iglesia, 2014; Gollapalli et al., 2015; Kenekayoro et al., 2014), which means that they cannot be used in a general context.

In this article, we present CALA, a tool that helps software engineers in generating URL patterns that allow to classify web pages without having to download them previously. Our tool does not suffer from the previous drawbacks: First, it does not require an extensive crawling of the site to build the classification model, but only a small subset of hub pages, which are automatically extracted from the web site by a lightweight crawler (hub pages are link-rich web pages that are returned when a web form is submitted). Furthermore, it is not supervised, since it does not require the hub pages to be labelled by the user. To fill in the forms and retrieve the hubs, we use keywords that can be extracted from the same site automatically, hence no dictionary or user input is needed. Furthermore, it is based exclusively on URL features, so pages do not have to be downloaded prior to be classified. Finally, it is site-, language-, and domain-independent, since it does not require any details regarding the site, the language or the domain that the site pages belong to in order to make it work properly. Therefore, our tool is both scalable and generally applicable. Furthermore, our tool provides a GUI that allows to gather the sets of hub pages from a particular web site, to generate the patterns that represent the different classes of pages in that site, and to evaluate the classification performance of the patterns in terms of precision, recall and F_1 score. Although we have identified some limitations of CALA when dealing with certain types of web sites, these limitations are easy to overcome, as we report on Section 5.2. We have validated our tools using a collection of 22 real-world web sites, from which we extracted 100 hubs. CALA never took more than 2 min to learn patterns, and they could be applied in a time that was negligible; furthermore, the F_1 score reveals that the precision and the recall that our tool can achieve are very high.

We have previously published some preliminary ideas (Hernández et al., 2012), a report on a preliminary version of our tool (Hernández et al., 2011), and a formalisation of the algorithms behind our tool (Hernández et al., 2014). In this article, we present the final architecture of our tool in detail, and we give an account of its implementation. Also, we present an algorithm to reconstruct the patterns, which are created without considering the URL separator tokens such as '/' and '?', and transform them into wildcarded URLs.

The rest of the article is organised as follows: Section 2 presents the related work; Section 3 reports on some preliminaries that are necessary to understand our tool; Section 4 describes the underlying data model and algorithms of CALA; Section 5 presents how we implemented our tool and the results of our experimentation; finally, Section 6 recaps on our main conclusions.

2. Related work

In this section, we present other existing tools that are related to CALA, since they focus on web page classification. We present some tools that are based on content-based features, others that use link-based features, and others that use URL-based features. Finally, we analyse and discuss the drawbacks of these tools.

2.1. Content-based tools

Content-based classification tools analyse features that can be computed from the contents of the web page to be classified; obviously, these tools require the pages to be downloaded before they can be classified. Content-based features for classification include the collection of terms of each page, its structure, and its visual aspect once rendered by a web browser.

Term-based classifiers usually represent web pages as vectors of terms, i.e., vectors containing the stems of the words that appear in the page, once the stop words have been removed. Some of these tools are based on the well-known bag of terms approach that is frequently used in text processing. According to it, a page is represented as a map from terms onto their frequency; later, a proposal to learn a classifier from these frequencies can be used, such as neural networks (Selamat and Omatu, 2004), k-nearest neighbours (Kwon and Lee, 2003), or support vector machines (Dumais and Chen, 2000). Some tools that use the term vector representation are FTC (Beil et al., 2002), COSA (Hotho et al., 2002), or WPCM (Selamat and Omatu, 2004).

Structure-based classifiers build on the idea of template. Web pages are usually generated by means of server-side templates that provide the structure of the pages and have placeholders that must be filled in with data by means of server-side scripts (Bar-Yossef and Rajagopalan, 2002). As a consequence, web pages that are generated by the same template are likely to belong to the same class; to classify a new web page it must be compared to the different templates and assigned to the class whose template is more similar. Some tools that are based on the page structure are TPM (Blanco et al., 2008), Local Template Detection (Bar-Yossef and Rajagopalan, 2002), RTDM (de Castro Reis et al., 2004), and the tool devised by Vieira et al. (2006), which is a variation from the latter.

Finally, visual-based classifiers use features that can only be computed when a web page is rendered by a browser, e.g., the position of an image on the screen, its bounding box, the distance to other elements in the page, and so on. Visual features are used to distinguish between different areas or vision blocks in a web page, which are sections of the page that represent a single unit with a certain functionality or topic. Then, the classification of web pages is based on the idea that pages that belong to the same class usually organise their vision blocks similarly. Therefore, to classify a web page, it is compared against other annotated pages, and assigned the class of the page that distributes its vision blocks more similarly. Visual-based classification is not a very researched area; the only related tool that we have found is the one devised by Zhu et al. (2008).

2.2. Link-based tools

Link-based tools classify web pages not only considering the web page itself, but also the links to and from other web pages. Usually, these tools represent the Web as a graph in which nodes are web pages, and edges are links from one web page to another.

Following this approach, Bhagat et al., 2007 proposed a semi-supervised link-based classifier that focuses on blogs, i.e., web pages in which users publish personal information about their lives and interests. It is based on the idea that people usually include in their blogs links to the blogs of other people with whom they share some common interests or demographical attributes (e.g., age, location, or gender). Their tool takes a web page graph in which some nodes are pre-classified as input, and uses the information provided by classified nodes to predict the classes of the other nodes. de Campos et al. (2008) explored the same idea to classify web pages (not only blogs) using a Bayesian network. Finally, some tools do not use link-based features in isolation, but together with other types of features to improve the classification. As an example, Getoor et al. (2001); Zhu et al. (2007) and Xie et al. (2007) included link-based features into a term-based web page classifier, which helped them achieve significant improvements regarding the precision of their classifier.

2.3. URL-based tools

A naïve approach to URL-based web page classification is to use clustering techniques; they rely on a distance function and return a number of clusters that verify that the inter-distance is maximum, whereas the intra-distance is minimum. Since URLs can be naturally represented as strings, the idea would be to use a string distance. Unfortunately, it has been noticed that using classic string distances does not work well to classify URLs (Blanco et al., 2011) since two close URLs may provide information about two different classes, whereas distant URLs may be related to web pages of the same class. For instance, there is a minimum distance between URLs[1] http://⟨MSAS⟩/Detail?entitytype=2&searchtype=2&id=35096884 and http://⟨MSAS⟩/Detail?entitytype=1&search-type=5&id=35096884, but they reference web pages that are likely to be classified in different classes (publications of an author and citations made to that author's papers). Contrarily, URLs like http://⟨MSAS⟩/Author/2542366/charles-antony-richard-hoare and http://⟨MSAS⟩/Author/10540585/yu-li are far more distant but belong to the same class (authors). It remains unexplored whether using non-classic distances might improve the results.

[1] For the sake of brevity, hereinafter, in our examples we use <MSAS> as an abbreviation for Microsoft Academic Search domain name, academic.research.microsoft.com.

Beyond the previous naïve approach, other authors have proposed a number of ad-hoc tools that are built on different techniques to classify web pages based on their URL.

Brin (1998) presented *DIPRE*, a supervised tool to extract structured information from web pages. It considers the Web as a database of unstructured information, and it aims at gathering tuples from it (e.g., books). This tool takes a set of sample tuples as input, and it performs an incremental process that consists of the following steps: first, it looks for occurrences of the sample tuples in the Web, i.e., it looks for web pages where the attributes of one of the tuples occur near to each other. For each occurrence, the URL of the web page on which it appears and the text that surrounds it are considered the context of the occurrence. Afterwards, *DIPRE* uses these contexts to generate patterns that match occurrences with a similar context. These patterns include a URL prefix, which is the longest common prefix to the URLs of the occurrences, and a text pattern, which is a regular expression that matches the text surrounding the occurrences. Finally, it looks for tuples in the Web matching the new patterns. The process iterates until enough patterns have been generated. Note that *DIPRE* requires performing an extensive crawling of the Web to gather as many tuples of the target relation as possible.

Shih and Karger (2004) proposed *Learn-RD* and *Learn-WW*, two unsupervised web page classification tools that are based on the idea that two visually nearby elements probably belong to the same class and, likewise, similar URLs probably have similar pages as target. Their tools tokenise a set of training URLs using the characters '/', '&', and '?' as separators, and insert the tokens in a tree structure. The root of the tree contains the first token (e.g., `http:`), and the other tokens are progressively inserted in order in the tree, each of them as a child of the previous tokens. Then, the tools build a Bayesian network from the tree as follows: each node in the tree is initially assigned a class so that the probability of a token belonging to the same class as the parent token is maximised; to prevent overfitting, a mutation probability that allows a child token to change its class is introduced. Then, some leaves in the tree are assigned the class they have in a trained set of annotated URLs, and the classes of other nodes in the tree are updated according to their mutation probability. Finally, each URL is assigned the class of its associated leaf node. Both tools are based on the same algorithm, but they differ on the URL training datasets. Their tools combine URL features with other features that are found inside a web page, such as the anchor text or the location of a link inside de page. Note that these features require the page to be downloaded beforehand.

Kan and Thi (2005) proposed a supervised web page classifier for pages in different web sites that is based exclusively on features computed from the tokenisation of their URLs. The URLs are tokenised using the standard RFC 3986 format for URIs; then, more features are computed, such as the position of each token in the URL, the length of the URL, or the lexical type of token (e.g., if it represents a number, a word, or a non-alphabetical symbol). These features are used as input to an entropy maximisation algorithm, a well-known machine learning approach that is usually applied to text classification (Berger et al., 1996; Nigam et al., 1999). To build the classifier, they use large training sets of URLs to achieve good precision and recall, which requires a previous extensive crawling of the sites that are being analysed.

Baykan et al. (2011) presented a supervised web page classification tool that creates feature vectors by tokenising URLs and then uses those features to build a support vector machine and a naïve–Bayes classifier. In their experiments, they use large training sets of URLs, and they require the user to provide a list of words and URLs that are representative of every class; furthermore, they also require a sample set of URLs that are not representative of each class.

Vidal et al. (2008) proposed *URLPattern*, a supervised tool to classify web pages. *URLPattern* takes a sample page as input, and returns a set of URL patterns that match the URLs of pages that are structurally similar to the sample page. It is based on two steps: site mapping and pattern generation. Site mapping consists in building a map of the web site, which requires to crawl the entire site starting from its home page and following every possible path. They keep a record of the paths in the map that lead (directly or indirectly) to pages that are similar to the sample page. The similarity is measured using a tree-edit distance between the DOM trees underlying the pages. Then, pattern generation consists of generalising the URLs of the pages in the former paths using regular expressions, and then selecting the path that leads to the largest number of target pages.

Bar-Yossef et al. (2009) proposed *DustBuster*, a supervised tool to detect web pages with different URLs that have the same contents, which has a negative impact on crawling efficiency. To solve this problem, they classify URLs according to the contents of their target, and they build regular expressions to define each class of URLs. Then, those URLs are normalised using a rule mining algorithm. They need to have a large collection of URLs to achieve good results, which means that a previous extensive crawling of the web site must be performed to gather them. A similar tool was proposed by Koppula et al. (2010).

Blanco et al. (2011) proposed *MDL-UC*, an unsupervised tool to classify web pages that combines URL-based features with optional content-based features. Their tool is based on the idea that every web site is created by populating a number of HTML templates with data from a database, and that the URLs of those pages are created by populating a URL template with data from the same database. Therefore, pages created from the same HTML template have similar contents and URLs generated from the same URL template link to pages with similar contents. They proposed an algorithm that combines web page contents and URL-based features to cluster web pages so that each cluster contains pages that were created using a certain template. Their tool is based on the well-known minimum description length method (Grünwald, 2005). They require a large training set, so they crawl the entire site in their experiments. Note that to improve the classification efficiency, content-based features can be used, which means that the page must be downloaded previously in some cases.

Abdallah and de la Iglesia (2014) proposed another supervised tool to classify URLs based on splitting them into n-grams, i.e., sequences of n characters. They create a language model for each class as the probability distribution of n-grams in the training set for that class. Then, to classify a URL u, it is first split into n-grams, and then assigned the class that maximises the probability value of its set of n-grams. This tool is based on the language model built from the training set, so the resulting classifier is highly dependent on the language used to create the URL. Moreover, it is based on a supervised technique, which means that it requires to retrieve a training set; however, the experimental results show that this training set does not have to be significantly large in order to achieve good classification results.

Kenekayoro et al. (2014) proposed a supervised tool to automatically classify academic web pages from the United Kingdom based on two machine learning techniques: decision tree induction and support vector machines. Its goal is to identify the potential relationship that exists between pages from different academic web sites that are connected through a hyperlink, by analysing the classes of both the referring and the referred page. To compose a training set of URLs, their tool relies on a crawler that starts at the homepages of a given set of universities web sites in the UK, and it performs an exhaustive crawling on those web sites to locate the existing links between their pages. The crawler stops visiting the links from a web site when it has visited 2000 web pages in that web site without finding any link. Their classifier uses two types of features: words from the page title and words from the page URL; therefore, this tool is site-, language-, and domain-dependent.

Gollapalli et al. (2015) proposed a semi-supervised tool that performs focused crawling to retrieve academic researcher homepages. This tool is built on a technique that combines term- and URL-based features to classify each web page visited by the crawler, in order to guide it towards finding the researcher homepages as soon as possible. Their tool is based on co-training, i.e., it requires two different sets of features (content-based and URL-based features), that provide complementary information about each example in the training set, and two sets of training examples, a small labelled set and a larger unlabelled set. One classifier is trained for each set of features using labelled examples. Then, both classifiers are used to classify unlabelled examples, and the most confident predictions of each classifier are added to the labelled examples set. Finally, this process is repeated iteratively until every example has been labelled. Since this tool is designed to find researcher homepages, it is domain dependent. Moreover, it requires URLs to be written using identifiable terms, which makes it site dependent. Finally, it is only able to deal with web sites in English. Therefore, it is site-, language-, and domain-dependent.

2.4. Discussion

Table 1 summarises the comparison of current tools to classify web pages. In this table the √ symbol denotes that the tool supports a feature, symbol × denotes that the tool does not support a feature, symbol N/A denotes that this feature is not applicable to the tool, and the ∼ symbol entails that the feature is partially supported. The features we have analysed are the following:

F_1: This feature determines if a tool is able to create a classification model based on a previous lightweight crawling, i.e., one that does not need to crawl a significantly large portion

Table 1
Comparison of current web classification tools. F_1 = Lightweight crawling; F_2 = unsupervision; F_3 = classify without downloading; F_4 = site independent; F_5 = language independent; F_6 = domain independent.

Tools	F_1	F_2	F_3	F_4	F_5	F_6
Content-based tools						
Beil et al. (2002)	×	√	×	√	√	√
Hotho et al. (2002)	√	∼	×	√	×	×
Kwon and Lee (2003)	×	×	×	√	√	√
Selamat and Omatu (2004)	×	×	×	√	√	√
Bar-Yossef and Rajagopalan (2002)	×	√	×	√	√	√
Blanco et al. (2008)	√	×	×	√	√	√
de Castro Reis et al. (2004)	×	√	×	√	√	√
Vieira et al. (2006)	×	√	×	√	√	√
Zhu et al. (2008)	√	×	×	√	√	√
Link-based tools						
Bhagat et al., 2007	×	∼	√	×	√	×
de Campos et al. (2008)	×	×	√	√	√	√
Getoor et al. (2001)	×	×	×	√	×	√
Zhu et al. (2007)	×	×	×	√	√	√
Xie et al. (2007)	×	×	×	√	√	√
URL-based tools						
Brin (1998)	×	×	√	√	√	√
Shih and Karger (2004)	√	×	∼	×	×	√
Kan and Thi (2005)	×	×	√	√	×	√
Vidal et al. (2008)	×	×	√	√	√	√
Bar-Yossef et al. (2009)	×	√	√	√	√	√
Koppula et al. (2010)	×	√	√	√	√	√
Baykan et al. (2011)	×	×	√	√	×	×
Blanco et al. (2011)	×	√	∼	√	√	√
Abdallah and de la Iglesia (2014)	∼	×	√	√	×	√
Gollapalli et al. (2015)	×	∼	×	×	×	×
Kenekayoro et al. (2014)	×	×	∼	×	×	×
CALA	√	√	√	√	√	√

of the collection of web pages in a web site to achieve good classification results.

F_2: This feature determines if a tool is unsupervised, i.e., if it is able to perform the classification without requiring any pre-classified web page sample from the user.

F_3: This feature determines whether or not a tool is able to classify a web page without downloading it previously.

F_4: This feature determines if a tool is site-independent, i.e., if it is not tailored to deal with just a number of specific web sites.

F_5: This feature determines if a tool is language-independent, i.e., if it is able to deal with web sites written in different languages.

F_6: This feature determines if a tool is domain-independent, i.e., if it is able to deal with web sites belonging to different domains.

These features have been selected since they allow assessing if a given classification tool can be successfully applied in real-world scenarios, which is our focus. To achieve this, the tools should be scalable, efficient, and generally applicable. The first requirement is neither fulfilled by tools that need to perform an extensive crawling to gather a training set, nor by supervised tools, since they are based on a handcrafted training set, which is a costly procedure; the second requirement is not fulfilled by tools that need to download a page before classifying it; the third requirement is not fulfilled by tools that are site-, language-, or domain-dependent.

Most of the tools that we have analysed, except for the ones by Shih and Karger (2004), Blanco et al. (2008), Hotho et al. (2002), Zhu et al. (2008), and Abdallah and de la Iglesia (2014), require a previous extensive crawling of the web site under analysis, to create a training dataset. Such a crawl could interfere with the normal operation of the site, which is not desirable. Furthermore, web sites change frequently (Fetterly et al., 2004; Koehler, 1999) and it is not uncommon that these changes render the classifier obsolete. Therefore, the classifier must be learnt not once but several times and performing an extensive crawling that covers a large portion of a web site becomes unfeasible. Contrarily, our tool does not require an extensive crawling of the site under analysis, but a lightweight crawling that retrieves a small number of pages from the site, which does not interfere with the normal operation of the site, and makes it scalable.

Regarding the degree of supervision, most of the analysed tools are supervised, which means that a person needs to pre-classify each page in the training set, which is an effort-consuming and error-prone task. This renders supervised tools unappealing for real-world web sites, since they shall not scale well to the Web (Madhavan et al., 2008). Contrarily, our tool is unsupervised; it is trained on an unlabelled set of pages that is automatically retrieved by a lightweight crawler, which saves time and human effort, and makes it scalable.

Regarding the type of features, the tools by Getoor et al. (2001), Xie et al. (2007), and Zhu et al. (2007) are link-based tools, but they require some content-based features to work well. The tools by Blanco et al. (2011), Gollapalli et al. (2015), Kenekayoro et al. (2014), and Shih and Karger (2004) are URL-based tools, but they also require some content-based features, which implies that they all require to download the pages to be classified. Downloading a web page puts a load on the server, takes time, and consumes bandwidth. That is, it is important that a classifier relies exclusively on external features of a page in order to classify it, since it would be inefficient for practical purposes otherwise (Kan and Thi, 2005). Contrarily, our tool uses exclusively URL-based features, which avoids having to download a page before classifying it.

Finally, some tools are either site-, language-, or domain-dependent (Shih and Karger, 2004; Kan and Thi, 2005; Baykan et al., 2011; Getoor et al., 2001; Bhagat et al., 2007; Hotho et al., 2002; Kenekayoro et al., 2014; Gollapalli et al., 2015; Abdallah and de la Iglesia, 2014), which means that they are not applicable to every site in the Web. Contrarily, our tool has been used to classify web pages in a number of different real-world sites belonging to diverse domains, and we report on our results in Section 5. Note that our analysis includes sites in English and German for evaluation purposes only, since we had to be able to identify the class of each page to evaluate the tool precision and recall; however, our tool is able to deal just as well with sites in different languages, since it does not rely on lists of words or thesauri.

3. Preliminaries

Next, we introduce some preliminary concepts that are necessary to understand our tool in Section 3.1, and the conceptual model on which our tool is based in Section 3.2.

3.1. Research methodology

Our research methodology builds on the Open Unified Process reference framework (Kruchten, 2003), which we have been using for several years, both for research and technology transfer. Within this general framework, the project is divided into the following phases:

1. Identifying research context: previous to this piece of research work, we identified that classifying web pages automatically was an interesting topic, and came to the conclusion that to classify a web page in the real-world context, it should not be necessary to download it beforehand. Therefore, we decided to focus on the URL-based web page classification.
2. Systematic review of the bibliography. We identified the existing tools and techniques to perform web page classification.
3. Identifying comparison features: we identified those features that are common to existing tools in our research context. These features are described in Section 2.4.
4. Identifying drawbacks: using the previous features, we analysed existing tools in the bibliography regarding whether they have these features or not. The conclusion was that, to the best of our knowledge, no tool has all of the features.
5. Design and implementation of our tool: we devised CALA to take all of the identified features into account.
6. Design of the experiments: every tool should be tested using real-world scenarios to evaluate its effectiveness and efficiency. We identified 22 real-world web sites amongst the most visited sites in the Web to test our tool (see Section 5.1).

3.2. Conceptual model

A URL is a sequence of characters that identifies a resource and describes its access protocol. The URL syntax was defined by the IETF in RFC 3986 (IETF). According to this recommendation, a URL is composed of different types of segments: first, a protocol (e.g., html, ftp, and so on); then, an authority or domain name (e.g., `academic.research.microsoft.com`); afterwards a sequence of path segments separated by slash characters (e.g., `/Detail`); and finally two optional sections: a question mark symbol followed by a query string, and/or a sharp symbol followed by a fragment. A query string is a structure that provides information about the names and the values of the parameters sent to the web server (e.g., `?entitytype=1&searchtype=5&id=48814179`), whereas the fragment is a sequence of characters that indicates a specific

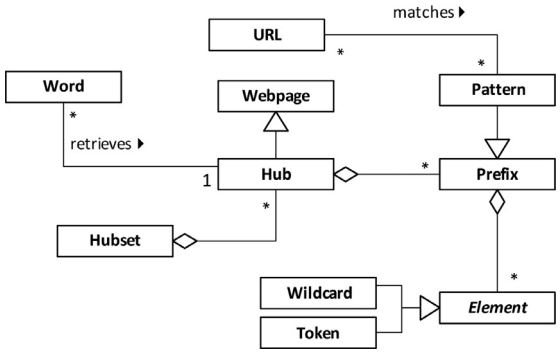


Fig. 1. Conceptual model.

section inside a page. We denote any of the subsequences of characters in a URL that is limited by separators as a token, where separators are '/', '?', '#', '&', '=' and ':'.

We define a pattern as a sequence of URL tokens and wildcards, that ends with a $ symbol. A wildcard, which we denote as $\star$, is a placeholder that accounts for any token. Note that given a URL, it is straightforward to transform it into a pattern; thus, we do not provide any additional details on this procedure. Also, since encoding special characters in a URL is a well-known procedure, we can safely assume that patterns do not include special characters.

A prefix refers to a subsequence of a pattern, that starts on the first token and extends up to any token in the pattern. Note that a pattern is similar to a prefix, since they both are sequences of tokens and wildcards; the only difference between them is that a pattern ends with $. We introduce both terms for the sake of clarity since prefix emphasises that it is a subsequence of the larger sequence that is the pattern. We formally define the previous concepts as follows:

$$
\begin{aligned}
&[URL]\\
&[Token]\\
&Prefix == \mathrm{seq}(Token \cup \{\star\})\\
&Pattern == \{p{:}Prefix \cdot p \frown \langle \$ \rangle\}
\end{aligned}
$$

A hub page is a special kind of web page that results from submitting a search form using some words as query, and provides summaries and links to other pages (Kleinberg, 1999). Note that hub pages usually contain a larger number of URLs than other pages in a web site since their goal is to offer the users as many results related to their queries as possible. Therefore, the probability that they contain a sufficiently representative set of URLs is higher than for other pages. Regarding our tool, a hub can be abstracted as a set of patterns that result from transforming the URLs in the links provided by that hub page. Note that we consider search forms that consist of a unique text field that is filled in with words. A hubset is a collection of hubs that result from submitting several times a search form using different words. We formally define the previous concepts as follows:

$$
\begin{aligned}
&[Word]\\
&[WebPage]\\
&Hub == \mathrm{set}Pattern\\
&Hubset == \mathrm{set}Hub
\end{aligned}
$$

Fig. 1 presents a UML-like conceptual model, in which a *Hubset* comprises a set of *Hubs*. Each *Hub* is a specific type of *Webpage* that is retrieved as a result of submitting a form using a *Word*. A *Hub* comprises a set of *Prefixes*, each of which is a sequence of elements that can be either *Tokens*, or *Wildcards*. A *Pattern* is a special type of *Prefix* that ends with a $ and matches a number of *URLs*.

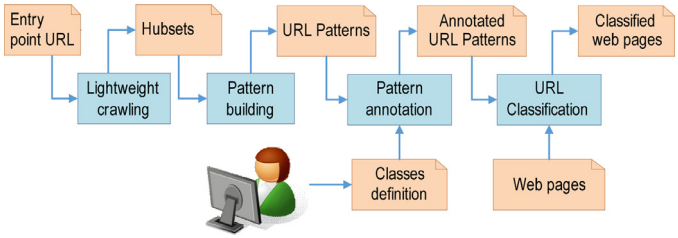


Fig. 2. Overview of our web page classification process.

4. CALA

Our tool takes the URL of the entry point to a web site, i.e., the URL of a web page that contains a keyword-based search form that can be filled and submitted in order to retrieve hub pages. Usually, the web site home page fulfils this requirement.

To illustrate the automated classification of web pages using CALA, we provide a running example that consists of classifying the web pages in the Microsoft Academic Search web site. It is an scholarly web site that offers information about items that include papers, authors, citations, and publishing hosts, such as journals or conferences. As an example, Fig. 3 displays the search form that constitutes the entry point to the <MSAS>web site.

Once the tool has retrieved a hubset, it builds a number of URL patterns that represent the different classes of URLs of that web site. It is expected that the user has to annotate those URL patterns a posteriori. Note that the size of the set of patterns is significantly smaller than the set of pages in a site, so the cost of annotating them is negligible.

Fig. 2 presents an overview of CALA, which comprises four steps, namely: (1) "Lightweight crawling" takes the URL of the entry point to a web site, and automatically gathers a set of hubs from that web site. (2) "Pattern building" uses the former set to build a set of patterns that represent the URLs in that site. (3) "Pattern annotation" relies on the user to annotate each URL pattern, by assigning them a semantic label. (4) "URL classification" uses the set of annotated patterns to classify new web pages from the same site by finding which pattern, if any, matches its URL. If no match is possible, this means that we have found a page whose URL deviates largely from the URLs from which we learned the patterns, which is likely to be due to a reorganisation of the web site. In such cases, it is necessary to learn the patterns again.

Of these four steps, step (3) is manual, although we have devised an ancillary tool to support the annotation process, by assigning different colours to the different URL patterns, and colouring the URLs that match each pattern with the corresponding colour. This makes it easier for the user to identify the class behind each pattern. Step (4) is trivial, since to classify a web page it suffices to compare the web page URL to the URL patterns to find the best match. Therefore, in this article, we focus on steps (1) and (2). However, we provide some details about our ancillary tool in Section 5.

Fig. 3. Search form in the running example.

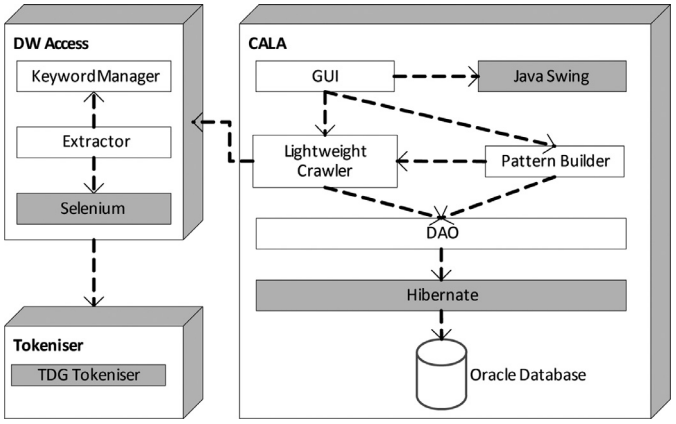


Fig. 4. Component diagram of CALA.

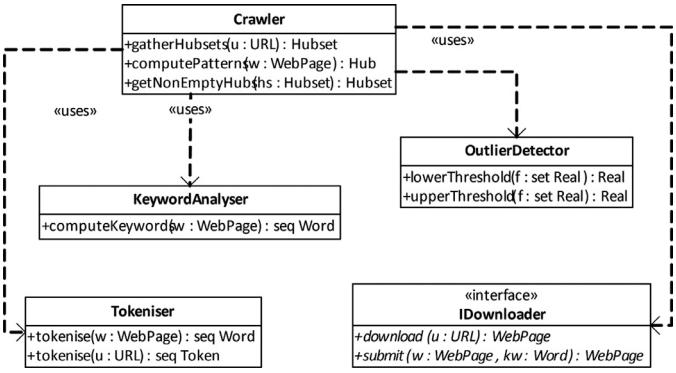


Fig. 5. Class diagram of the crawler.

Fig. 4 presents the architecture of our tool. Our tool comprises three components: first, the main CALA component, that includes the lightweight crawler, URL pattern builder and the graphical interface, all of which is supported by an Oracle database. CALA also includes a DAO layer to deal with the persistence. Then, the Deep Web access component that handles the automatised access to the Web, form filling and submission, and managing the extracted keywords. Finally, the Tokeniser, that is responsible for parsing URLs and transforming them into sequences of tokens. This component is based on the TDG Tokeniser, devised by some members of our research group (Sleiman and Corchuelo, 2013).

In the next subsections, we first describe the lightweight crawler in Section 4.1; then, we describe the pattern builder in Section 4.2. Finally, we discuss on the implementation of our tool in Section 5.

4.1. Lightweight crawler

The architecture of the crawler is presented in Fig. 5. Class *Crawler*, which provides function *gatherHubsets* to gather a set of hubs, has to interact with a keyword-based search form. Therefore, it needs a number of keywords, which are provided by class *KeywordAnalyser*. This class analyses the pages of a web site to gather the keywords by tokenising their pages into words. This tokenisation is provided by class *Tokeniser*. The interaction with the web site is performed by interface *IDownloader*, which is responsible for handling the HTML requests. Finally, an outlier detection technique is needed to discard empty hubs, i.e., hubs that do not contain any result relevant to the query, which is provided by class *OutlierDetector*.

Function *gatherHubsets* orchestrates the other elements of the architecture. Fig. 6 presents the main algorithm that implements

```
 1: algorithm gatherHubsets
 2: input     fp : URL
 3: output    Result : Hubset
 4: variables wp : WebPage; n : ℕ; P, Q : set Word
 5:
 6: n := 0
 7: - Step 1: Download initial page
 8: wp := download(fp)
 9: - Step 2: Compute keywords from page contents
10: Q := computeKeywords(wp)
11: P := ∅
12: while n ≤ T ∧ Q ≠ ∅ ∧ #Result < M do
13:    while Q ≠ ∅ ∧ #Result < M do
14:       - Step 3: Submit form using keyword from Q
15:       kw := get one keyword from Q
16:       P := P ∪ {kw}
17:       wp := submit(fp, kw)
18:       - Step 4: Create hub using the URLs in wp
19:       Result := Result ∪ {computePatterns(wp)}
20:       - Step 5: Update the set of keywords
21:       Q := Q ∪ computeKeywords(wp) \ P
22:    end while
23:    - Step 6: Keep only non-empty hubs
24:    Result := getNonEmptyHubs(Result)
25:    n := n + 1
26: end while
```

Fig. 6. Algorithm to gather hubs starting from a form page of a web site.

Table 2
Summary of the global constants used by our algorithms and the values that we suggest for them in the illustrating example.

Constant	Suggested value
M	100
T	5
N	20
α	0.05

such function. It takes the URL of a page with a keyword-based search form with at least one text field as input and outputs a hubset. First, the downloader downloads the page and the keyword analyser chooses the words with a lower frequency in the page as keywords that can be used to issue queries from the search form. Then, the downloader finds a text field in the search form, fills it with the keywords and submits the form, which yields some hub pages as a response. These pages are processed twofold: on one hand, the crawler processes them to compute the patterns that they contain and to create hubs; on the other hand, they are used to compute more keywords. Finally, the crawler applies an outlier detection technique based on the well-known Cantelli inequality (Mallows and Richter, 1969) to discard empty hubs. This process is repeated until enough hubs have been retrieved. The result is a hubset that contains a representative collection of URLs from the web site. We assume that the following constants have been set before executing this algorithm: M, which refers to the number of hubs the algorithm is expected to return; T, which refers to the maximum number of attempts that the algorithm is allowed to make in order to gather M hubs; N, which refers to the number of keywords that we select from each page; and α, which determines the fraction of elements in a distribution that are considered outliers when applying the Cantelli inequality. Table 2 provides a

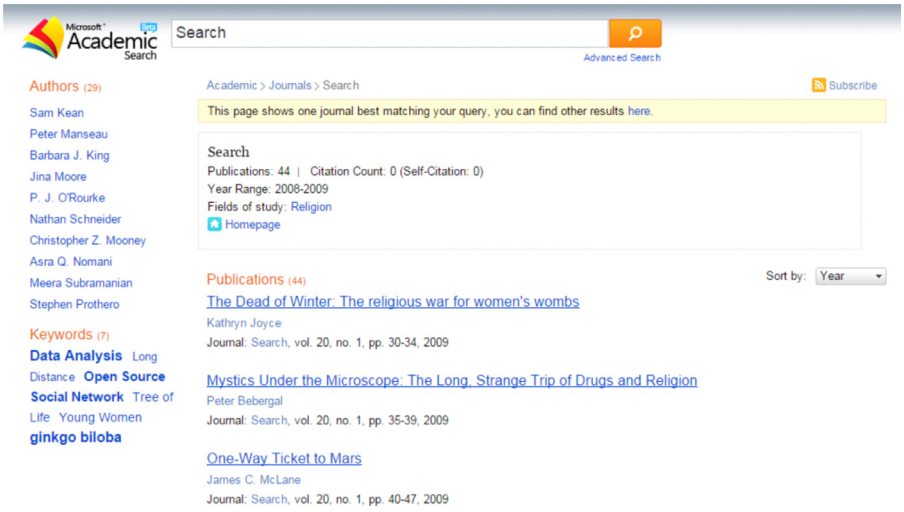


Fig. 7. Hub in the running example, retrieved after submitting the form with keyword *Search*.

summary of the constants that are used by our algorithms and the values we suggest for them.

In some cases, a keyword may not retrieve any result, returning an empty hub. Since empty hubs are not useful for our purposes, function *getNonEmptyHubs* discards those hubs that are likely to be empty, by applying an statistical approach. In the worst case, a web site may not return any non-empty hub, in which case, *getNonEmptyHubs* returns an empty hubset. In the next iteration, more empty hubs are gathered and later discarded. If the algorithm keeps finding keywords that yield empty hubs, it may never stop. Therefore, we introduce a parameter T that represents the maximum number of attempts. If the size of the hubset after T iterations is not large enough to be useful for the pattern builder, then we must discard that web site.

This algorithm is supported by some functions that are not defined in this article due to lack of space: *download*(u), which downloads a web page; *computeKeywords*(*webPage*), which tokenises a web page and extracts the list of keywords in the page content (excluding HTML tags); *submit*(*webPage, kw*), which locates an existing search form inside *webPage*, fills in the form using keyword *kw*, submits it, and gathers another web page as a result (note that *submit* makes an inner call to *download*); and finally, *computePatterns*(*webPage*), which composes a Hub by extracting all of the patterns (URLs) inside *webPage*.

Example 1. To illustrate this step, we assume that the constants in our algorithm have been set to the values suggested in Table 2. The input variable *fp* takes the URL of the <MSAS> home page; therefore, the algorithm first downloads it and then retrieves the page that contains the search form that we presented in Fig. 3. Then, a maximum of 20 keywords are extracted from the page content, namely: *Search*, *Organisations*, *Fields*, *Publications*, *Business*, *Authors*, *Study*, *Science*, *Mathematics*, *Advanced*, *Economics*, *Geosciences*, *Arts*, *Humanities*, *Sign*, *Journals*, and *Keywords*. They are used to fill in and submit the form. This results in a collection of hubs; for instance, Fig. 7 shows the hub that is retrieved using keyword *Search*.

Further keywords are extracted from these hubs and are used to fill in the form and to retrieve more hubs. The process continues until at least 100 hubs have been retrieved, or until the crawler runs out of keywords. Some of the hubs might be empty, i.e., the web server does not have information related to some particular keywords, and the result of submitting the forms with these keywords is an informative page with no results. Fig. 8 depicts an empty hub from our running example web site.

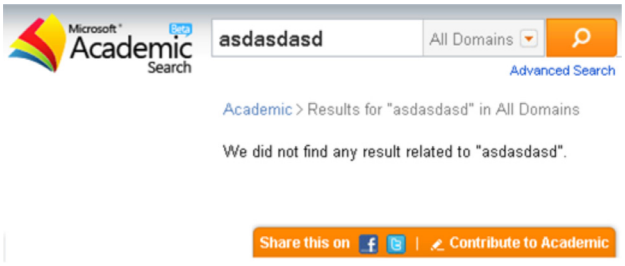


Fig. 8. Empty hub in the running example.

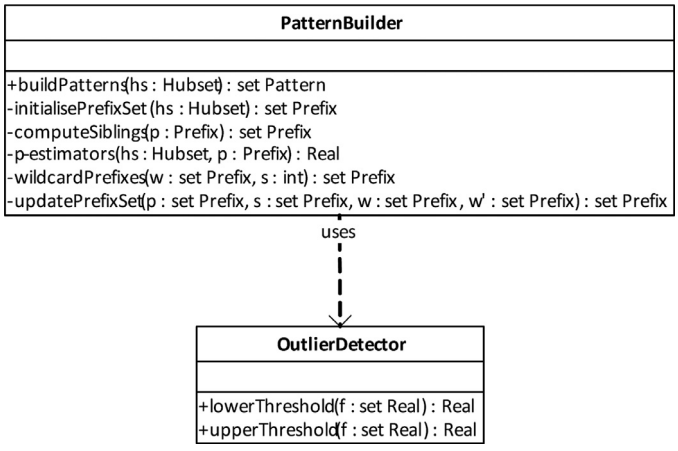


Fig. 9. Class diagram of the pattern builder.

After discarding empty hubs, less than 100 may remain; in that case, it is necessary to try again with the form submission and keyword extraction, until the minimum number of hubs has been reached, or the crawler has ran out of keywords. This process is repeated a maximum of 5 times; after that, if the crawler has been unable to retrieve 100 hubs, it stops.

4.2. Building URL patterns

The architecture of the pattern builder is presented in Fig. 9. Class *PatternBuilder* provides the functions needed to build a set of patterns. We apply an outlier-based statistical technique to build the patterns. Therefore, an outlier detection method is needed,

```
algorithm buildPatterns
input      hs : Hubset
output     P' : set Pattern
variables  P, S, NS, W, W' : set Prefix; p, q : Prefix; pe, t : ℝ

- Step 1: P contains every possible prefix in hs
P := initialisePrefixSet(hs)
P' := ∅
while P ≠ ∅ do
  p := shortest prefix in P
  if last p = '$' then
     P := P \ {p}
     P' := P' ∪ {p}
  else
     - Step 2: compute siblings
     S := computeSiblings(hs, p)
     - Step 3: compute threshold
     t := upperThreshold({q : Prefix | q ∈ S
 · p-estimators(hs, q)})
     NS := ∅
     for each q ∈ S do
         - Step 4: compute probability estimator
         pe := p-estimators(hs, q)
         if pe ≤ t ∧ pe ≉ 1.0 then
             NS := NS ∪ {q}
         end if
     end for
     - Step 5: prefixes are wildcarded
     W := {v, w : Prefix | v ∈ NS ∧ w ∈ P ∧ v prefix w · w}
     W' := wildcardPrefixes(W, #p)
     - Step 6: update prefix set
     P := updatePrefixSet(P, W, W', S, NS)
  end if
end while
```

Fig. 10. Algorithm to build URL patterns.

which is provided by class *OutlierDetector*. This outlier detector is based on the same outlier detection technique used by the crawler (cf. Section 4.1).

Fig. 10 shows our algorithm that implements function *buildPatterns*. It takes a hubset *hs* gathered by the crawler as input and outputs a set of patterns. First, it extracts the set of prefixes of size greater than 1 for every pattern in every hub from *hs* ordered by length. Then, it iterates on the set of prefixes *P* until it is empty. In each iteration, the algorithm selects the shortest prefix in *P*. If the selected prefix ends in $, which marks the end of a pattern, it means that all prefixes in the pattern have already been processed, and it can be added to the output set of patterns; otherwise, the prefix has to be processed along with its siblings, i.e., the prefixes that share a common prefix with it, excluding its last token. Then, the algorithm computes their probability estimators building on their frequencies and calculates the threshold above which a p-estimator is considered an upper outlier using the outlier detection technique. Later, prefixes with a probability estimator smaller than a threshold are wildcarded, by replacing the token that occupies their last position with a wildcard. Finally, prefix set *P* is updated by withdrawing the prefixes that have already been processed in this iteration and adding those of their descendants that have been wildcarded. The result is a set of URL patterns that represent the URLs of the different classes of pages in the web site.

```
algorithm  buildRegularExpression
input      p : Pattern, m : Pattern ↠ seqSeparator
output     result : URL
variables  i : ℕ

result := ""
for i ∈ 1..#p
     result := result · p(i) · m(i)
end for
```

Fig. 11. Algorithm to build a regular expression from a pattern.

We assume that constant α has been set to a proper value before executing this algorithm.

This algorithm is supported by some functions that are not defined in this article due to lack of space: function *initialisePrefixSet*(*hs*) returns a sequence with the prefixes of the patterns in flat*hs*, ordered in increasing size order; function *computeSiblings*(*hs, p*) returns the set of prefixes in flat*hs* that share a common prefix with prefix *p* up to its penultimate element, including *p* itself; function *p-estimators*(*hs, q*) returns an estimator of the probability of finding at least one pattern prefixed by prefix *q* in hubset *hs*; function *wildcardPrefixes*(*W, i*) returns a set in which the input prefixes in set *W* have been wildcarded at the *i*th position, i.e., the token at this position has been changed into a wildcard; finally, function *updatePrefixSet*(*P, W, W', S, NS*) returns a prefix set that is the result of updating *P* by replacing the prefixes to be wildcarded with their wildcarded version in each iteration of the algorithm, and subtracting the sibling prefixes that have been processed in that iteration.

Example 2. In our running example, after processing the hubset we get the following patterns:

$p_1 = \langle$`http`, `<MSAS>`, `Publication`, $\star$, $\star\rangle$,
$p_2 = \langle$`http`, `<MSAS>`, `Author`, $\star$, $\star\rangle$,
$p_3 = \langle$`http`, `<MSAS>`, `Journal`, $\star$, $\star\rangle$, and
$p_4 = \langle$`http`, `<MSAS>`, `Detail`, `entityType`, `1`, `searchType`, `5`, `id`, $\star\rangle$.

Pattern p_1 matches the URLs of pages with information about papers, i.e., it matches web pages belonging to semantic class `Paper`, pattern p_2 matches pages of class `Author`, pattern p_3 matches pages of class `Journal` and p_4 matches pages of class `Citation` in Microsoft Academic Search.

Note that the patterns output by Algorithm *buildPatterns*, do not contain URL separators, only tokens and wildcards. However, actual URLs do contain separators; to use the patterns to classify URLs it is then necessary to reconstruct the patterns and transform them into wildcarded URLs. Therefore, when the URLs are tokenised to create the initial patterns in function *computePatterns*(*webPage*), a map *m* is built that associates each pattern to the sequence of separators between each pair of tokens. This map is used to build the former regular expression before using the patterns to classify web pages, cf. algorithm in Fig. 11.

Example 3. In our running example, after building the regular expressions associated to each pattern, we get the following:

$p_1 = \hat{}$`http://<MSAS>/Publication/`$\star$`/`$\star$`$`,
$p_2 = \hat{}$`http://<MSAS>/Author/`$\star$`/`$\star$`$`,
$p_3 = \hat{}$`http://<MSAS>/Journal/`$\star$`/`$\star$`$`, and
$p_4 = \hat{}$`http://<MSAS>/Detail/?entityType=1&searchType=5&id=`$\star$`$`.

5. Evaluation

Our tool is supported by a graphical interface that has been implemented using Java 1.6.0_25. Furthermore, we have used

Guava 13.0.1 to implement ancillary set operations, Hibernate and Oracle for persistency, and HTMLCleaner 2.1 to parse the HTML code of the web pages and produce well-formed versions of the pages, which is essential for the evaluation process. Also, we used GNU-Regex for manipulating the regular expressions on which our URL patterns are based. Furthermore, since our evaluation is based on XPath, we used Xalan and a Firefox plug-in (Slesinsky, 2012) to identify XPaths of the links that led to each of the classes that we selected for every site.

In our implementation, we use an interface for the crawler downloader to abstract away from the technology. We tested two different implementations for interface *IDownloader* using Selenium and WebDriver. WebDriver (Stewart and Burns, 2013; Selenium, 2012) belongs to the Selenium project (Holmes and Kellogg, 2006), but instead of injecting Javascript functions in the browsers to execute the scripts, it calls their native APIs. An additional difference is that WebDriver executes the scripts without a user interface.

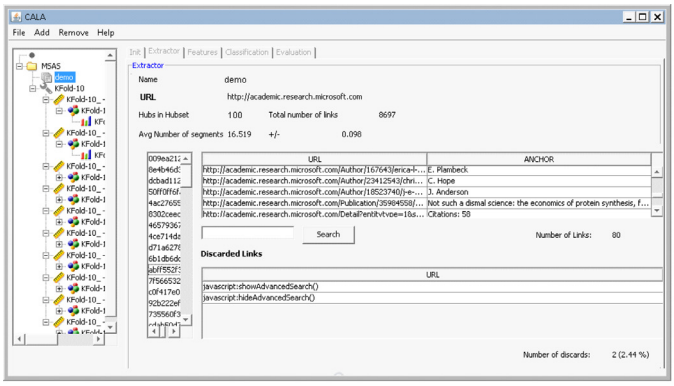
Our tool has four modules, namely: Lightweight crawling, Pattern building, URL classification, and Evaluation.

In the Lightweight crawling module (cf. Fig. 12a), the user can visualise the hubsets that have already been extracted from web sites, remove them, or extract a new hubset by specifying a name, URL of the web page that contains the form that acts as an entry point of the web site, and the values for the parameters that tune the performance of the crawler, namely: the number of parallel threads, whether the extractor should be invisible to the user, and constants M, N, and T. Once a hubset has been extracted, the user is able to visualise the list of hubs that compose it. When the user clicks on any of them, CALA displays the complete set of URLs that have been extracted from the links in each hub and the anchor text of those links. Furthermore, CALA displays a listing of links that have been discarded because they do not lead to HTML pages (e.g., links to JavaScript functions, FTP server locations, or mailto: links).

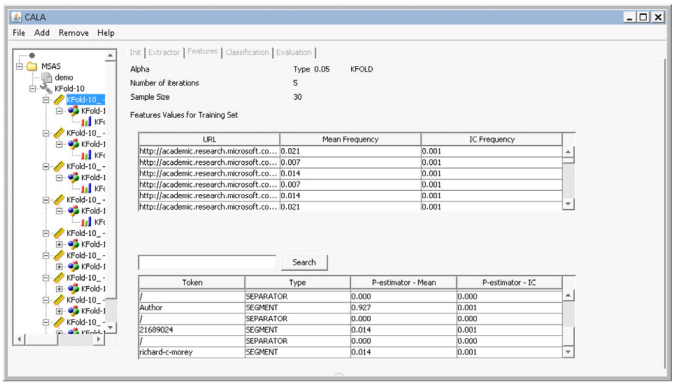
The Features module (cf. Fig. 12b) allows the user to visualise the p-estimators that have already been calculated in previous executions, as well as to calculate a new set of p-estimators for a given hubset specifying the value for the α constant. The user may as well choose to discard some of the URLs if they belong to the template of the web site, i.e., if they are found in a high percentage of hubs from the hubset. This might be useful, for example, if the built URL patterns are used for guiding a crawler to look exclusively for pages with data that are relevant to a query, in which case any URL that appears in almost every page in the site, regardless of the query submitted, is likely not useful for the crawler. Finally, the user may choose to discard minor URL patterns, i.e., those that match a significantly low number of URLs in the site.

The Classification module (cf. Fig. 12c) allows the user to visualise the information about the URL patterns that have been built by the tool from a particular hubset, namely: their associated regular expression and their coverage. The user may choose to manually select URL patterns to discard, besides those that might have been discarded because of their low coverage. Furthermore, the user may combine two or more URL patterns, in which case the tool combines their regular expressions into one. Finally, the user may choose to evaluate the classification performance of the set of URL patterns, which activates the Evaluation module.

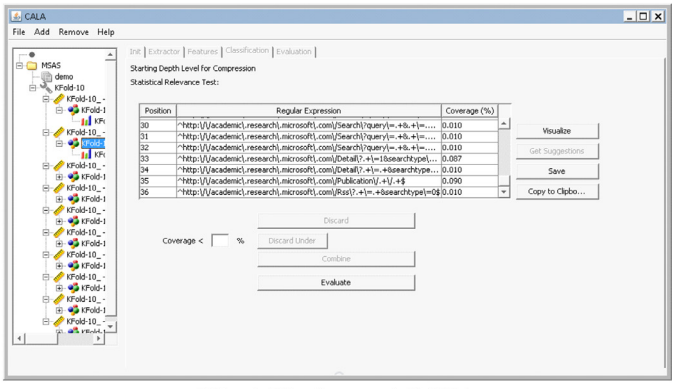
Finally, the Evaluation module (cf. Fig. 12d) allows the user to evaluate the classification performance of a particular set of URL patterns, in terms of precision, recall and F_1 score. To do so, the user may choose between two evaluation modes: supervised and unsupervised. In the supervised mode, the user is able to define a number of classes of URLs and to specify correspondences between the classes and the URL patterns. In the unsupervised mode,



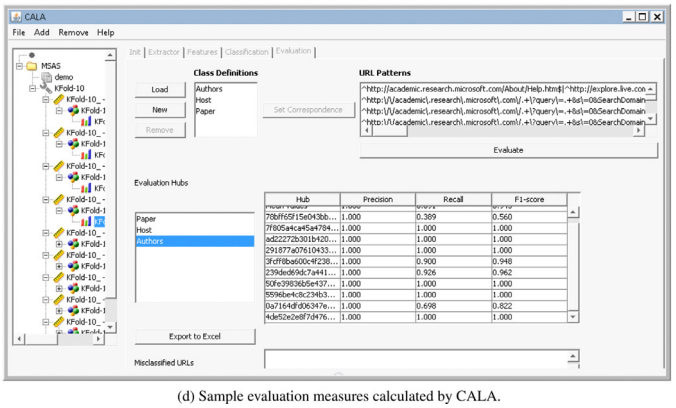
(a) Sample hubset retrieved by CALA lightweight crawler.



(b) Sample p-estimators calculated in CALA.



(c) Sample URL patterns generated in CALA.



(d) Sample evaluation measures calculated by CALA.

Fig. 12. Screen shots from our tool.

Table 3
Results of the evaluation. P = Precision; R = Recall; F_1 = F_1 -score; LT = CPU learning time; HDT = hubset download time; MCT = mean classification time, MDT = mean page download time. The timings are expressed in seconds.

Site	*URLs*	*Tokens/URL*	*P*	*R*	F_1	*LT*	*HDT*	*MCT*	*MDT*
Amazon	30,749	31.11	1.00	0.98	0.99	2.78	16.52	0.08	0.87
Anwers	13,840	12.00	0.96	1.00	0.98	2.06	20.38	0.09	0.50
Arxiv	33,748	23.29	1.00	0.96	0.98	5.42	41.06	0.09	0.12
Battle.net	4845	14.13	1.00	1.00	1.00	9.48	22.16	0.09	0.23
BBC	8096	15.38	1.00	0.79	0.87	8.86	56.30	0.09	0.22
Chip	16,698	14.27	1.00	1.00	1.00	7.97	13.77	0.10	0.36
DailyMail	30,300	21.06	1.00	0.73	0.81	3.73	24.91	0.10	0.64
DailyMotion	5039	15.21	1.00	1.00	1.00	8.02	13.63	0.10	0.13
Deviantart	14,262	13.86	0.97	0.66	0.74	2.08	11.86	0.10	0.48
Drupal	9700	9.10	1.00	0.84	0.91	20.94	11.23	0.10	0.20
Filestube	12,951	17.55	1.00	0.94	0.97	7.59	57.30	0.10	0.06
Fotolia	20,887	16.83	1.00	1.00	1.00	21.55	15.19	0.10	0.74
Indeed	9306	15.06	1.00	1.00	1.00	5.28	13.50	0.11	0.21
Livejournal	10,456	10.88	1.00	0.64	0.78	2.08	41.66	0.11	0.52
MsAcademic	6734	16.57	1.00	0.92	0.96	3.41	28.75	0.02	0.50
Netlog	7333	16.39	1.00	1.00	1.00	4.47	18.56	0.11	0.47
Newegg	47,464	28.03	1.00	1.00	1.00	6.16	20.78	0.12	1.34
Odesk	10,853	14.18	1.00	1.00	1.00	3.00	50.42	0.12	0.73
People	7811	19.56	1.00	0.81	0.83	3.14	23.84	0.12	0.63
Slideshare	5402	16.44	0.92	1.00	0.96	0.66	85.08	0.12	0.37
Squidoo	6192	12.77	1.00	1.00	1.00	1.95	4.63	0.12	0.53
Torrentz	9704	9.27	1.00	1.00	1.00	4.25	14.34	0.12	0.48

the user does not define correspondences, but only classes. In this mode, CALA applies a technique that is based on the proposal by Marxer et al. (2007) to calculate precision and recall in a non-supervised fashion; that is, not assuming an a-priori correspondence between each URL pattern and one of the classes. Instead, intermediate precision and recall are calculated for every possible combination of pattern/class, and the final precision and recall are the weighted means of the intermediate values, where the weight is the number of URLs that the pattern and class of each combination have in common. By doing so, we are taking into account the possibility that our patterns include URLs from more than one class of pages, and vice versa.

Since CALA is based on an unsupervised technique, a person must interpret the patterns that it outputs. The size of the set of patterns is significantly smaller than the set of pages in a site, so the cost of interpreting them is expected to be negligible. Furthermore, to support the assignment of a class to each pattern, we have developed an ancillary tool for graphically displaying URL patterns. It displays the URLs that match each pattern on a sample web page using different colours, so it helps to identify the class behind each pattern. An algorithm chooses the colour palette so that any two given colours are significatively different from each other. Therefore, it is easy to distinguish between the URLs that match two given URL patterns. A demo of this tool is available at the author's web page.[2] Fig. 13 displays a screen shot of our ancillary tool.

Note that module Lightweight crawling supports the Lightweight crawling step in our workflow (cf. Fig. 2), modules Features and Classification support the Pattern Building step, and our ancillary tool supports the Pattern annotation step.

Our experiments were run on a cloud computer that was equipped with a four-threaded 64-bit 2.93 GHz Intel i7 processor, 16GB of RAM, running on Windows 7 Pro 64-bit. In the rest of this section, we present the effectiveness and efficiency evaluation (cf. Section 5.1) and the limitations of our tool (cf. Section 5.2).

[2] http://www.tdg-seville.info/inmahernandez/CALA+Demo.

Fig. 13. Sample web page coloured with our ancillary tool.

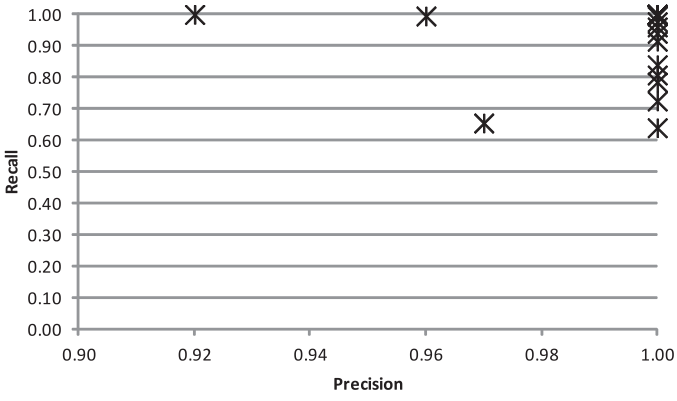


Fig. 14. Scatter plot showing the precision versus recall of CALA.

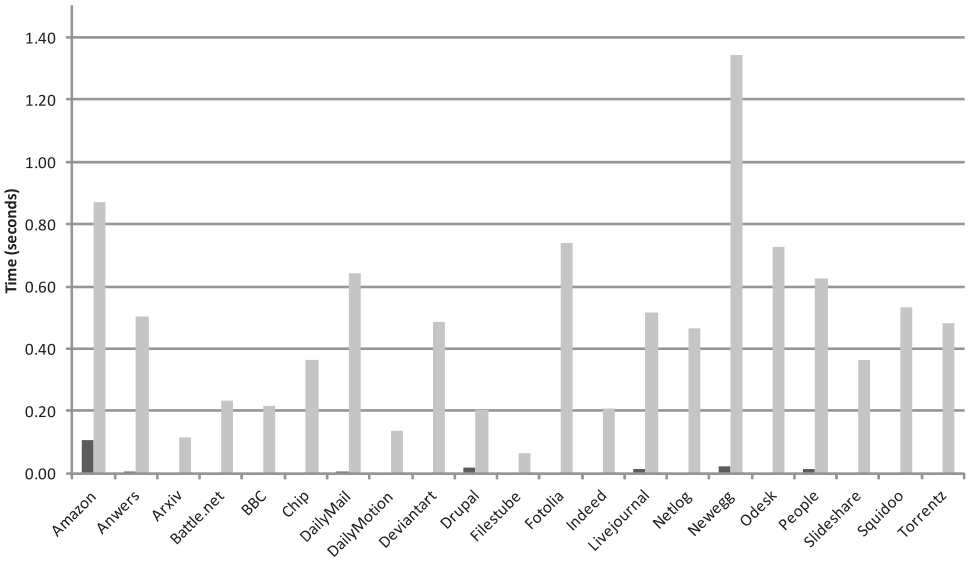


Fig. 15. Comparison between the average download time for a regular web page (light grey bars) and the average time that CALA takes to classify it (dark grey bars).

5.1. Effectiveness and efficiency evaluation

Repository. We have setup a repository of 22 real-world hubsets; 20 of them were chosen from the Alexa top 500 web sites that were written in English or German and provided a keyword-based search form. Since the Alexa ranking (Alexa, 2012) changes daily, we fixed a reference date to select the sites (February 14, 2011). The dataset included two additional academic sites, namely: Arxiv, and our running example Microsoft Academic Search. From each site, we downloaded 100 hubs to perform the experiments, i.e., we had 2200 pages available for experimentation.

Evaluation process. For each web site, we identified the classes that best described their pages. For instance, in Amazon, we identified the following classes: `Product`, `Author`, and `Review`. Since we had 2200 pages, we decided to use a technique to automate the labelling of the pages: for each hub, we used XPath-Checker to identify the XPaths of the links that led to each of the classes that we selected for every site. The labelling took roughly 50 work hours, whereas we estimated that labelling each page individually would have taken more than 1, 000 work hours, not to mention that this would have led to many classification errors and would have been difficult to scale. The XPath expressions, the dataset with the retrieved hubsets, and the lists of keywords that were retrieved from each web site is available for downloading at the author's web site.[3]

For the configuration of our crawler, we set the parameters to the values suggested in Table 2. Other things equal, increasing *M, T*, or *N* did not have an impact on the effectiveness of our tool; however, decreasing *M, T*, or *N* had a negative impact when analysing a few sites. Regarding α, we selected the most commonly used value in the literature.

For each dataset, we used our pattern builder and evaluated its effectiveness. We used 50% of the hubs in each dataset to infer a set of patterns (training set) and the remaining 50% to validate our tool (test set).

Evaluation results. Table 3 reports on the results of our experiments. The columns report on the number of URLs extracted from each site (*URLs*), mean number of tokens per URL (*Tokens*/*URL*), precision (*P*), recall (*R*), F_1 score ($F_1 = 2\frac{PR}{P+R}$), CPU learning time in seconds (*LT*), time to download each hubset in seconds (*HDT*), mean time taken by CALA to classify a web page using the URL patterns in seconds (*MCT*), and mean time spent downloading a web page from the site in seconds (*MDT*).

Regarding effectiveness, our main conclusion is that CALA achieves good classification results, as indicated by the F_1 score values; the scatter plot in Fig. 14 illustrates this conclusion since the majority of points that correspond to CALA are very close to the upper right corner ($P \geq 0.90$, $R \geq 0.90$). Note that 71.43% of the data points from CALA are located in this corner.

Regarding efficiency, the CPU learning times range from 0.66 to 20.94 s, while the hubset download times range from 4.63 to 85.08 s, which means that the total time spent creating the patterns never exceeds 2 min. This makes CALA quite appealing for real-world web page classification problems. Note that this timings correspond exclusively to the Lightweight crawling and URL pattern building steps, which only have to be executed once. After the patterns have been built, the mean time required to classify each page is negligible in almost every case, and significantly lower than one second in every case. To compare the efficiency of our tool to other classifiers that are based on content-based features and require to previously download the page, we provide the mean time required to download a page in column *MDT*. Just the *MDT* is already significantly higher than CALA *MCT*, at least eight times higher in the best case, as illustrated in Fig. 15; this confirms the hypothesis that URL-based classification is significantly more efficient than content-based classification. We provide a comprehensive experimental evaluation of our tool and a comparison of the efficiency and effectiveness of our tool to other tools in the state of the art elsewhere (Hernández et al., 2014).

5.2. Limitations

Currently, our tool can only deal with web sites that provide a keyword-based search form. The reason for this is that hubs are rich in links, and they allow to gather a set of URLs that are representative from the whole set of URLs in the web site, just by downloading a significantly small number of web pages. This is actually a soft limitation that does not hinder the practical applicability of our tool, since nowadays the majority of web sites provide a keyword-based search form. In those rare cases in which a web site does not provide one, this limitation can be overcome. Algorithm *buildPatterns* works with any set of pages from the web site, although its results depend on those pages

[3] http://www.tdg-seville.info/inmahernandez/Experiment-JSS.

providing a representative set of URLs. Therefore, to deal with web sites that do not provide a keyword-based search form, Algorithm *gatherHubsets* can be modified to gather a set of pages from the web site by other means, e.g., by using a traditional blind crawler (Raghavan and Garcia-Molina, 2001). In this case, the set of pages should provide CALA with a representative set of URLs in order to build effective URL patterns.

Another limitation of our tool is that it does not work well with shortened URLs, which are short versions of URLs that have been very used lately, especially in the context of social media. The small length of these URLs and the absence of significant terms makes it difficult for CALA to create URL patterns for a particular site. As an example, consider a typical URL from our running example site: http://academic.research.microsoft.com/Author/5201962/carlos-r-rivero. After applying a shortening service such as TinyURL, the resulting URL is http://tinyurl.com/o2qxovm, which has lost the structure and information of the original URL. However, this limitation can be easily overcome, since shortened URLs are merely a translation of longer and structured URLs, and our tool works well with the latter.

6. Conclusions

Web page classification is an interesting area that has been extensively researched because of its many applications. Unfortunately, most of the existing tools to automatically classify web pages are not appealing for real world web sites due to a number of drawbacks, namely: they require a previous extensive crawling, which is not suitable for large web sites; they are supervised, which has a negative impact on their scalability; they are based on content-based features, which means that a web page has to be downloaded before classifying it, which hinders their efficiency; or they are either site-, language-, or domain-dependent, which makes them non-generally applicable.

In this article, we present CALA, a tool to automatically generate URL-based web page classifiers. Our tool takes the URL of a web page with a keyword-based search form as input, and it outputs a set of patterns that represent the URLs of pages that belong to each semantic class. It builds on a statistical outlier detection technique to decide which parts of a URL are significantly representative and which parts can be abstracted. The strongest features of our tool are that it does not require a previous extensive crawling, it is unsupervised, it is able to classify a web page without actually downloading it, and it is site-, language-, and domain-independent.

We have validated our tool using a collection of datasets that were gathered from 22 real-world web sites that we have made publicly available. We have built a set of patterns for each web site, and we have used them to classify further pages. Our validation results show that CALA is very effective in practice, as indicated by the F_1 score values that it achieved. The times required to build those patterns were reasonable and appealing for real-world scenarios, since they never exceeded 2 min. Furthermore, the classification times were negligible in almost every case; in the worst cases, they were always significantly lower than one second. These results suggest that our tool seems promising enough for real-world web page classification, that it is efficient in practice, and that the patterns it builds are able to classify web pages accurately, which makes it suitable for a real-world web page classification scenario.

Acknowledgements

Our work was supported by FEDER funds that were awarded by the European Commission, the Spanish Government, and the Andalusian government through contracts TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, TIN2011-15497-E, and TIN2013-40848-R.

References

Abdallah, T.A., de la Iglesia, B., 2014. URL-based web page classification – a new method for URL-based web page classification using n-gram language models. In: Proceedings of KDIR, pp. 14–21. doi:10.5220/0005030500140021.

Alexa, 2012. Alexa top 500 sites ranking. http://www.alexa.com/topsites (accessed 1.03.12).

Bar-Yossef, Z., Keidar, I., Schonfeld, U., 2009. Do not crawl in the dust: different URLs with similar text. Trans. Web 3 (1), 3.

Bar-Yossef, Z., Rajagopalan, S., 2002. Template detection via data mining and its applications. In: Proceedings of International Conference on World Wide Web, WWW, pp. 580–591.

Baykan, E., Henzinger, M., Marian, L., Weber, I., 2011. A comprehensive study of features and algorithms for URL-based topic classification. Trans. Web 5, 15:1–15:29.

Baykan, E., Henzinger, M.R., Marian, L., Weber, I., 2009. Purely URL-based topic classification. In: Proceedings of International Conference on World Wide Web, WWW, pp. 1109–1110.

Beil, F., Ester, M., Xu, X., 2002. Frequent term-based text clustering. In: Proceedings of International Conference on Knowledge Discovery and Data Mining, KDD, pp. 436–442. doi:10.1145/775047.775110.

Berger, A.L., Pietra, S.D., Pietra, V.J.D., 1996. A maximum entropy approach to natural language processing. Comput. Linguist. 22 (1), 39–71.

Bhagat, S., Cormode, G., Rozenbaum, I., 2007. Applying link-based classification to label blogs. In: Proceedings of International Conference on WebKDD/SNA-KDD, pp. 97–117. doi:10.1007/978-3-642-00528-2_6.

Blanco, L., Crescenzi, V., Merialdo, P., 2008. Structure and semantics of Data-IntensiveWeb pages: an experimental study on their relationships. J. UCS 14 (11), 1877–1892.

Blanco, L., Dalvi, N., Machanavajjhala, A., 2011. Highly efficient algorithms for structural clustering of large websites. In: Proceedings of international conference on World Wide Web, WWW. ACM, pp. 437–446.

Brin, S., 1998. Extracting patterns and relations from the World Wide Web. In: Proceedings of International Conference on WebDB, pp. 172–183.

de Campos, L.M., Fernández-Luna, J.M., Huete, J.F., Romero, A.E., 2008. Probabilistic methods for link-based classification. In: Proceedings of International Conference on INEX, pp. 453–459. doi:10.1007/978-3-642-03761-0_47.

de Castro Reis, D., Golgher, P.B., da Silva, A.S., Laender, A.H.F., 2004. Automatic web news extraction using tree edit distance. In: Proceedings of international conference on World Wide Web, WWW, pp. 502–511. doi:10.1145/988672.988740.

Chakrabarti, S., van den Berg, M., Dom, B., 1999. Focused crawling: a new approach to topic-specific web resource discovery. Comput. Netw. 31 (11–16), 1623–1640. doi:10.1016/S1389-1286(99)00052-3.

Crescenzi, V., Mecca, G., Merialdo, P., 2001. RoadRunner: towards automatic data extraction from large web sites. In: Proceedings of International Conference on VLDB, pp. 109–118.

Deza, M.M., Deza, E., 2012. Encyclopedia of Distances, 3rd ed. Springer.

Dumais, S., Chen, H., 2000. Hierarchical classification of web content. In: Proceedings of ACM International Conference on SIGIR, pp. 256–263. doi:10.1145/345508.345593.

Fersini, E., Messina, E., Archetti, F., 2008. Enhancing web page classification through image-block importance analysis. Inf. Process. Manag. 44 (4), 1431–1447. doi:10.1016/j.ipm.2007.11.003.

Fetterly, D., Manasse, M., Najork, M., Wiener, J.L., 2004. A large-scale study of the evolution of web pages. Softw.: Pract. Exp. 34 (2), 213–237. doi:10.1002/spe.577.

Getoor, L., Segal, E., Taskar, B., Koller, D., 2001. Probabilistic models of text and link structure for hypertext classification. In: Proceedings of IJCAI Workshop on Text Learning: Beyond Supervision, pp. 321–374.

Gollapalli, S.D., Caragea, C., Mitra, P., Giles, C.L., 2015. Improving researcher homepage classification with unlabeled data. ACM Trans. Web 9 (4), 17:1–17:32. doi:10.1145/2767135.

Grünwald, P., 2005. Advances in Minimum Description Length: Theory and Applications. MIT Press.

Hernández, I., Rivero, C., Ruiz, D., Corchuelo, R., 2011. A tool for link-based web page classification. In: Advances in Artificial Intelligence, Volume 7023 of Lecture Notes in Computer Science. Springer, Berlin/Heidelberg, pp. 443–452.

Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R., 2012. A statistical approach to URL-based web page clustering. In: Proceedings of the 21st International Conference Companion on World Wide Web. ACM, pp. 525–526. doi:10.1145/2187980.2188109.

Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R., 2014. CALA: an unsupervised URL-based web page classification system. Knowledge-Based Syst. 57, 168–180. doi:10.1016/j.knosys.2013.12.019.

Holmes, A., Kellogg, M., 2006. Automating functional tests using selenium. In: Proceedings of International Conference on AGILE, pp. 270–275.

Hotho, A., Maedche, A., Staab, S., 2002. Ontology-based text document clustering. Künstliche Intell. 16 (4), 48–54.

IETF, RFC 3986. https://www.ietf.org/rfc/rfc3986.txt (accessed 20.01.16).

Jain, A.K., Dubes, R.C., 1988. Algorithms for Clustering Data. Prentice-Hall.

Kan, M.-Y., Thi, H.O.N., 2005. Fast webpage classification using URL features. In: Proceedings of International Conference on Information and Knowledge Management, CIKM, pp. 325–326.

Kenekayoro, P., Buckley, K., Thelwall, M., 2014. Automatic classification of academic web page types. Scientometrics 101 (2), 1015–1026. doi:10.1007/s11192-014-1292-9.
Kleinberg, J.M., 1999. Authoritative sources in a hyperlinked environment. J. ACM 46 (5), 604–632.
Koehler, W., 1999. An analysis of web page and web site constancy and permanence. J. Am. Soc. Inf. Sci. 50 (2), 162–180. doi:10.1002/(SICI)1097-4571(1999)50:2⟨162::AID-ASI7⟩3.0.CO;2-B.
Koppula, H.S., Leela, K.P., Agarwal, A., Chitrapura, K.P., Garg, S., Sasturkar, A., 2010. Learning URL patterns for webpage de-duplication. In: Proceedings of International Conference on WSDM. ACM, pp. 381–390.
Kovacevic, M., Diligenti, M., Gori, M., Milutinovic, V.M., 2002. Recognition of common areas in a web page using visual information: a possible application in a page classification. In: Proceedings of IEEE International Conference on Data Mining, pp. 250–257.
Kruchten, P., 2003. The Rational Unified Process: An Introduction. Addison-Wesley Professional.
Kwon, O.-W., Lee, J.-H., 2003. Text categorization based on *k*-nearest neighbor approach for web site classification. Inf. Process. Manag. 39 (1), 25–44. doi:10.1016/S0306-4573(02)00022-5.
Li, Y., Zhong, N., 2004. Web mining model and its applications for information gathering. Knowledge-Based Syst. 17 (5–6), 207–217.
Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.Y., 2008. Google's deep web crawl. Proceedings of VLDB 1 (2), 1241–1252.
Mallows, C.L., Richter, D., 1969. Inequalities of Chebyshev type involving conditional expectations. Ann. Math. Stat. 40 (6), pp.1922–1932.
Marxer, R., Holonowicz, P., Purwins, H., Hazan, A., 2007. Dynamical hierarchical self-organization of harmonic, motivic, and pitch categories. In: Proceedings of NIPS Music, Brain and Cognition Workshop, Vancouver, Canada.
Nigam, K., Mccallum, A.K., Thrun, S., Mitchell, T., 1999. Text classification from labeled and unlabeled documents using EM. Mach. Learn. 39 (2–3), 103–134.
Raghavan, S., Garcia-Molina, H., 2001. Crawling the hidden web. In: Proceedings of International Conference on World Wide Web, WWW.
Selamat, A., Omatu, S., 2004. Web page feature selection and classification using neural networks. Inf. Sci. 158, 69–88. doi:10.1016/j.ins.2003.03.003.
Selenium, 2012. Webdriver. http://www.seleniumhq.org/projects/webdriver/ (accessed 2.04.12).
Shen, D., Chen, Z., Yang, Q., Zeng, H.-J., Zhang, B., Ma, W.-Y., Lu, Y., 2004. Web-page classification through summarization. In: Proceedings of International Conference on SIGIR, pp. 242–249. doi:10.1145/1008992.1009035.
Shih, L.K., Karger, D.R., 2004. Using URLs and table layout for web classification tasks. In: Proceedings of International Conference on World Wide Web, WWW, pp. 193–202.
Sleiman, H.A., Corchuelo, R., 2013. Tex: an efficient and effective unsupervised web information extractor. Knowledge-Based Syst. 39, 109–123. http://dx.doi.org/10.1016/j.knosys.2012.10.009.
Slesinsky, B., 2012. Xpath checker. https://addons.mozilla.org/en-US/firefox/addon/xpath-checker/ (accessed 1.03.12).
Stewart, S., Burns, D., 2013. WebDriver. Working Draft. World Wide Web Consortium.
Vidal, M.L.A., da Silva, A.S., de Moura, E.S., Cavalcanti, J.M.B., 2008. Structure-based crawling in the Hidden Web. J. UCS 14 (11), 1857–1876.
Vieira, K., da Silva, A.S., Pinto, N., de Moura, E.S., Cavalcanti, J.M.B., Freire, J., 2006. A fast and robust method for web page template detection and removal. In: Proceedings of International Conference on Information and Knowledge Management, CIKM, pp. 258–267. doi:10.1145/1183614.1183654.
Xie, W., Mammadov, M.A., Yearwood, J., 2007. Using links to aid web classification. In: Proceedings of International Conference on ACIS-ICIS, pp. 981–986. doi:10.1109/ICIS.2007.191.
Xu, R., Wunsch, D.C., 2005. Survey of clustering algorithms. IEEE Trans. Neural Netw. 16 (3), 645–678. doi:10.1109/TNN.2005.845141.
Xu, S., Yoon, H.-J., Tourassi, G., 2014. A user-oriented web crawler for selectively acquiring online content in e-health research. Bioinformatics 30 (1), 104–114.
Zhang, J., Qin, J., Yan, Q., 2006. The role of URLs in objectionable web content categorization. In: Proceedings of International Conference on Web Intelligence, pp. 277–283.
Zhu, M., Hu, W., Wu, O., Li, X., Zhang, X., 2008. User oriented link function classification. In: Proceedings of International Conference on World Wide Web, WWW, pp. 1191–1192. doi:10.1145/1367497.1367721.
Zhu, S., Yu, K., Chi, Y., Gong, Y., 2007. Combining content and link for classification using matrix factorization. In: Proceedings of International Conference on Research and Development in Information Retrieval, pp. 487–494. doi:10.1145/1277741.1277825.