BlogBuster: A tool for extracting corpora from the blogosphere

Georgios Petasis[1,2], Dimitrios Petasis[1]
[1]Intellitech S.A.
P.O. BOX 8055, GR-19300, Aspropirgos, Greece.
E-mail: petasisg@yahoo.gr, d_petasis@hotmail.com

[2]Software and Knowledge Engineering Laboratory,
Institute of Informatics and Telecommunications,
National Centre for Scientific Research (N.C.S.R.) "Demokritos",
P.O. BOX 60228, Aghia Paraskevi,
GR-153 10, Athens, Greece.
E-mail: petasis@iit.demokritos.gr

Abstract

This paper presents BlogBuster, a tool for extracting a corpus from the blogosphere. The topic of cleaning arbitrary web pages with the goal of extracting a corpus from web data, suitable for linguistic and language technology research and development, has attracted significant research interest recently. Several general purpose approaches for removing boilerplate have been presented in the literature; however the blogosphere poses additional requirements, such as a finer control over the extracted textual segments in order to accurately identify important elements, i.e. individual blog posts, titles, posting dates or comments. BlogBuster tries to provide such additional details along with boilerplate removal, following a rule-based approach. A small set of rules were manually constructed by observing a limited set of blogs from the Blogger and Wordpress hosting platforms. These rules operate on the DOM tree of an HTML page, as constructed by a popular browser, Mozilla Firefox. Evaluation results suggest that BlogBuster is very accurate when extracting corpora from blogs hosted in the Blogger and Wordpress, while exhibiting a reasonable precision when applied to blogs not hosted in these two popular blogging platforms.

1. Introduction

The huge amount of the available information found in the World Wide Web has been very attractive to researchers in natural language processing, especially in practical areas dominated by statistical and machine learning based approaches, where access to large volumes of data is highly desirable. The traditional way of constructing textual corpora, despite the fact that it is a time consuming and expensive process targeting corpora quality, may be inadequate to fulfil the current requirements of applications, especially the ones targeting the WWW. However, extracting corpora from the WWW is not an easy process, posing various difficulties in almost all steps of the extraction process as it is evident by the relevant research area represented by conferences such as Web as Corpus (WAC4, 2008; WAC5, 2009) and competitions such as CLEANEVAL (2007).

In general, the process of extracting a corpus from the WWW can be decomposed into the following major tasks:

1. Web crawling: the task of guided or automated browsing of the web, in order to identify web pages that should be downloaded and included into the corpus.
2. Web page download: the task of downloading the web page from the hosting web server, ensuring that the proper encoding conversions are applied so as the saved copy of the web page does not contain invalid data.
3. Web page cleaning: the task of extracting the desired elements from the web page, usually extracting the textual data from an HTML web page.

Several approaches have been presented in the literature that try to extract a corpus from the web, ranging from stripping the HTML elements from HTML web pages, to complex systems that employ machine learning to induce generic wrappers that identify and extract text blocks, i.e. (Spousta et. al., 2008). Most of these approaches usually apply heuristics to deal with problems such as malformed HTML code, or revert to specialised tools that try to disambiguate errors in HTML, such as HTML Tidy (Raggett, 2000) or CyberNeko HTML Parser (Clark and Guillemot, 2002), that usually convert HTML into a normalised form known as XHTML (W3C, 2000).

However, downloading a web page and converting it to text may not be enough for constructing a corpus. Since the HTML standard was designed mainly as visual description language, a web page may contain parts that exist only for providing a pleasant visual layout to the reader. In addition, the vast majority of current web sites contain modules that provide advertisements, site navigation links and other unwanted information that must be removed before extracting the page content. The process of identifying the blocks or parts of an HTML web page that contains useful information is known as page segmentation, and is usually a necessary task when aiming to construct a corpus that satisfies some quality requirements. Since efficiency of page segmentation

largely depends on the visual layout of the web sites, the difficulty of this task usually depends on the number of sources that needs to be handled. Several approaches have been proposed for the task, from site-specific wrapper induction, to machine learning approaches (Spousta et. al., 2008) or even approaches that combine semantic information, such as the distribution of named-entities (Petasis et. al., 2008).

The rest of the paper is organised as follows: section 2 discusses work related to the presented approach, while section 3 presents the requirements and motivation behind BlogBuster. Section 4 presents the extraction engine, followed by an evaluation presented in sections 5 and 6. Finally, section 7 concludes and outlines plans for future research.

2. Related work

The CLEANEVAL-1 (2007) competition targeted the topic of cleaning arbitrary web pages, with the goal of extracting a corpus suitable for linguistic and language technology research and development from web data. Providing a gold standard corpus for the English and Chinese languages, participant systems were evaluated and compared. The participant systems mainly employ machine learning algorithms to classify HTML fragments into various categories, based on a wide variety of features. For example, NCleaner (Evert, 2008) uses simple character-level n-gram language models in order to distinguish between clean body text and boilerplate (navigation bars, page headers, link lists, disclaimers, copyright statements, advertisements, etc.). Victor (Spousta et. al., 2008) employs a sequence-labelling approach, based on Conditional Random Fields, where every block of text is classified into *content* and *noisy* segments, using a set of features extracted from the textual content and HTML structure of an HTML document. Similarly, in (Kohlschütter et. al., 2010) decision trees are applied on shallow text features in order to classify the individual text elements of an HTML page. However, the use of machine learning does not necessarily ensure high performance: The BTE tool (Ferraresi et. al., 2008), which outperformed all CLEANEVAL-1 participants in the text-only task of the competition, uses heuristics based on the observation that the content-rich section of an HTML page has a low HTML tag density, while boilerplate exhibits a much higher density of HTML tags.

Other approaches try to exploit structural information, mainly contained in the DOM (Document Object Model) tree of an HTML page, in order to identify common, i.e. frequently used patterns or segments, usually concentrated on specific web sites. (Chakrabarti et. al., 2007) try to determine the "templateness" of DOM nodes, by training Logistic regression classifiers over a set o features, such as closeness to the margins of an HTML page, number of links per word, fraction of text within anchors, size of anchors, etc. At a second level, smoothing is applied on the classifier scores of DOM nodes using regularised isotonic regression to improve performance. Yi et. al. (2003) extract a "Site Style Tree" from HTML pages of a web site that tries to distinguish DOM nodes that contain actual content from presentation styles.

Despite the fact that many approaches for boilerplate removal exist, a greater level of detail may be required when dealing with the blogosphere. Removing the boilerplate may not be enough, as usually individual blog posts may need to be identified in the index page of a blog, along with the title and date of each blog post. Similarly, in an HTML page containing a single blog post, the identification of individual comments may also be desirable. This increased level of detail in extracting textual segments from blogs along with the scarcity of approaches that can provide this level of detail, was the driving motivation for BlogBuster, the tool described in this paper.

3. Requirements

Blogosphere is a web source that is characterised by the visual layout diversity, as each blog uses its own visual layout and theme. Such diversity will pose even more problems to existing page segmentation approaches, as there are too many layouts and themes that must be considered. Even commercial services that specialise into mirroring and organising the blogosphere, such as spinn3r (2007), do not attempt to perform the task of segmenting blog pages and extract information such as the blog post title, the blog post date and the blog post text, elements essential in order to construct a corpus from blogs.

The presented tool –BlogBuster– specialises in extracting a corpus from blogs, and in many aspects of the task follows a different approach, partly because it specialises only in processing the blogosphere. The requirements during the design of BlogBuster can be summarised into the following ones:

1. Cross-platform: the tool must not be bind to a specific operating system.
2. Robust: the tool must be as robust as possible both in its results, but also in the handling of invalid HTML code and dynamic content.
3. Cover as much part of the blogosphere as possible with adequate efficiency, without the efficiency being affected by the visual layout and theme diversity.

In order to fulfil these requirements, the following decisions were made with respect to the development of the tool:

1. The tool must be integrated with a cross-platform and popular web browser.
2. The engine that extracts the various pieces of information must operate over the DOM tree of the blog page, as constructed by the browser.
3. The extraction engine must be easily configurable and adaptable to various application requirements. In case of inadequate performance, application of heuristics that improve performance must be an easy task.

Mozilla's rendering engine (Gecko[1]) was chosen as a basis for BlogBuster. Despite the complexity in using this rendering engine, it was chosen because it is compliant to WWW standards and widely adopted (being the base for applications like Firefox, Thunderbird, Camino, Flock, SeaMonkey, K-Meleon, Netscape-9 and many others). The usage of such infrastructure as a basis provides many advantages over existing approaches:

1. Adequate handling of invalid or malformed HTML pages. Instead of relying on heuristics of HTML Tidy or NecoHTML, the correction facilities of a popular web browser are used. In addition, it is highly probable that blog authors do not allow errors in their blogs that will cause rendering errors in widespread browsers.
2. Ability to download and store locally a blog web page, by applying the proper character encoding conversions.
3. Ability to retrieve and store content that is generated dynamically (i.e. as a result of Javascript execution). Such content is constantly gaining in popularity and will be missed by any existing approach that does not understand and execute Javascript.

The BlogBuster tool performs the following actions upon receiving a URL of a blog (either the index/front page of a blog, or any other page containing a single blog post) to be processed:

1. Instructs the rendering engine to download and render the URL. The rendering engine downloads and applies style-sheet information, creating a visual representation of the web page identical to what a web browser would provide.
2. Instructs the rendering engine to execute any available Javascript code that must be executed.
3. The text extraction engine of the BlogBuster tool is invoked to operate upon the constructed by the rendering engine DOM (Document Object Model) tree. This extraction engine is responsible for identifying suitable DOM nodes that contain the required information, and extract the information from these nodes.
4. Collect the information extracted by the extraction engine, encode the results in the requested format (i.e. XML, JSON, plain text, etc.) and return the information to the caller.

4. The extraction engine

BlogBuster's extraction engine is implemented in the dynamic language Tcl/Tk[2], allowing for rapid prototyping and easy adaptation to various applications. The current application targets the extraction of individual blog posts, blog post titles and blog post publication dates from an HTML blog page, either the front page of a blog or a page containing a single post. However, the implementation of the extraction engine is generic enough to account for processing of additional sources, such as forums or portals. BlogBuster's text extraction from blogs has been implemented on the assumption that the vast majority of blogs are generated (and usually hosted) by only two very popular content management systems (CMSs): the Blogger [3] platform provided by Google, and the WordPress [4] CMS. Based on this assumption, the implemented extraction engine ignores all information that depends on visual layout and style/theme, and instead concentrates on the metadata stored by the CMSs on DOM node elements, like element ids and classes.

The current extraction engine operates on a small set of heuristics/rules, manually constructed by examining a limited set of HTML blog pages (~30 from both the Blogger and Wordpress platforms). The construction of these rules was based on the following observations:

1. The usage of the HTML tag is extremely widespread for the definition of structure and visual layout in blogs.
2. A element can be found that encloses the text of a single blog post.
3. A element can be found that encloses a single blog post comment (for blog posts that have comments).
4. All blog posts have a title. Titles are frequently contained inside a separate element (but not always).
5. All blog posts do not have a date. Availability and location of dates with respect to a blog post seem to depend on the visual layout and the theme of the blog, along with a decision of the blog owner to display a date. Dates, when available, may be incomplete (i.e. the year or month may be omitted). Dates are not usually contained inside a element that contains only the date.

The thematic domain of the observed blogs was related to blogs that comment on the news, written in the English language. More information about how these blogs were collected can be found in the section of experimental setting. The algorithm of the implemented extraction engine can be described as follows:

1. Collect all elements from the DOM tree of a blog HTML page. For each such element, identify whether it contains a blog post or not.
    a. A element is considered as containing a blog post if any of its attributes matches any of the following regular expressions: "post*", "*hentry*", "divlog", "*entry*post*", "*post-*", "snap_preview".
2. Extract the text from each element identified as containing a blog post.
    a. The text of a DOM node is

[1] http://en.wikipedia.org/wiki/Gecko_(layout_engine).

[2] Tcl/Tk is an open source language, distributed under the BSD license. More information can be found at http://www.tcl.tk.

[3] https://www.blogger.com/start

[4] http://wordpress.com/

extracted if the class of the node matches any of the following regular expressions: "*post-body*", "posts", "entry", "content", "*entry-content*", "snap_preview".

b. Else, try to locate a element node that is a child of the node under examination, whose class matches any of the following regular expressions: "*post-body*", "posts", "entry*", "content", "*entry-content*", "text", "entrybody". If a suitable node can be located, the text of the node is extracted and returned.

c. In all other cases, the node under examination is assumed to not contain a blog post.

3. For each DOM node that text was extracted, try to identify the title of the blog post.
   a. Examine all children nodes of type <h3>, <h2>, <a> (in this order). If any node attribute matches any of "*post-title*", "*entry-title*", or "*title*", the node is assumed to contain a title and its text is extracted and returned.
   b. Else, examine all previous sibling nodes, sorted by their proximity to the node under examination. If a sibling node is of type <h1>, <h2>, <h3>, <h4>, and <h5>, extract the sibling node text and return it as the title.
   c. In all other cases, declare a missing title for the blog post.
4. For each DOM node that text was extracted, try to identify the date of the blog post.
   a. If a child node can be found, whose class matches any of "*posted-in*", "datepost", "entrymeta", or "date", extract the text and return it as the posting date of the blog post.
   b. If a child node can found, whose class matches any of "postinfo", "info", or "*-head":
      i. If a child node can found, with an attribute matching "postdate" or "date", extract the text and return it as the date of the blog post.
      ii. If a child <small> node can be found, with an attribute matching "postdate", "date", "*time*", or "*date*", extract the text and return it as the date of the blog post.
   c. Else, examine all previous sibling nodes, sorted by their proximity to the node under examination. If a sibling node is of type <h1>, <h2>, <h3>, <h4>, <h5>, , <small>, and has an attribute *value* that matches "*date-header*", or "*date*", extract the sibling node text and return it as the date of the blog post.
   d. In all other cases, declare a missing date for the blog post.

As it is evident from the algorithm, segmenting a blog HTML page (either containing multiple posts or a single blog post) is a fairly easy process, while locating the title and especially the date is a more challenging task. In addition, the algorithm does not make any assumptions regarding the actual content and does not use any features from it, allowing operation over languages other than English. The algorithm has been tested on blogs written in French, German and Greek, obtaining results similar to those of blogs written in English. The algorithm mainly assumes that elements are used to construct the structure of pages, and that these elements contain attributes whose names roughly describe the type of the contained content, a feature quite frequent in CMSs that are designed to separate the content from the visual appearance, which can be customized through templates and styles. Finally the algorithm does not deal with comments that can be found in blog posts, which are ignored (not contained in the text extracted from blog HTML pages). However, the analysis of the limited set of blog pages during rule construction suggests that comments can be handled by the described approach.

5. Experimental setting

BlogBuster's extraction engine was manually constructed from a limited set of about 30 blogs (originating from blogs hosted in the Blogger and Wordpress platforms), and it was evaluated on the front pages from 416 blogs. The evaluation corpus was collected using the following approach[5]:

1. A small set of news agency portals were selected (i.e. www.usatoday.com, www.nytimes.com, www.fox.com, www.reuters.com, www.cnn.com, www.bbc.co.uk, etc.)
2. A small set of news items (~500) were collected from these sites, concerning news about sports, technology, and world politics.
3. Search keywords were extracted from the collected news items, by extracting all words from the titles of news items, along with distinctive words identified through TF/IDF.
4. Google's web and blog search was used to collect blogs for each news item, from both Blogger and WordPress. For each news item and hosting platform, the first 64 results were kept.

[5] The corpus collection approach targeted in creating a parallel corpus from news and blogs, with blog posts commenting on the news events contained in the HTML pages collected from the news agencies.

5. Technorati's [6] search was also used to collect blogs for each news item, without any restriction on the hosting platform.

Since blogs from the Blogger and Wordpress platforms have been used during the development of the extraction engine, blogs from these two platforms were randomly removed to reduce their number, in order to minimize the bias of the evaluation corpus towards these two platforms. No blogs used during the development of the extraction engine were present in the evaluation corpus. The characteristics of the evaluation corpus are shown in the following table:

Total number of blogs	416
Blogs from Blogger (*.blogspot.com)	49 (12%)
Blogs from Wordpress (*.wordpress.com)	28 (7 %)

Table 1: Characteristics of the evaluation corpus

6. Evaluation Results

The BlogBuster corpus extraction tool was evaluated on blog front pages (containing multiple blog posts) collected from 416 blogs. Several experiments have been contacted in order to evaluate various aspects of the approach, including the performance in extracting single blog posts, titles of blog posts and posting dates, for blogs hosted in the Blogger or Wordpress platforms, but also on blogs not hosted by these two platforms. The performance of BlogBuster was evaluated in terms of recall and precision.

The first evaluation was performed on blogs hosted by the Google's Blogger hosting platform. The obtained results are shown in the following table:

	Precision	Recall	F-Measure
Blog posts	100.00 %	85.71 %	92.30 %
Title	97.62 %	83.67 %	90.11 %
Date	90.48 %	77.55 %	83.52 %

Table 2: Performance on blogs from Blogger.

BlogBuster exhibits a good performance when extracting corpora from blogs hosted in the Blogger platform, mainly because the rules of the extraction engine originate from observations performed on this CMS. Precision of locating individual blog posts inside a blog HTML page containing multiple blog posts is high due to the rules used by the extraction engine: a single blog post is the central item detected by the extraction algorithm, before the extraction of other details (such as titles or dates) is attempted. In addition, the rules try to match only a small set of regular expressions over the attributes of DOM nodes that can contain the text of a single blog post, providing a bias towards precision instead of favoring recall. Performance for titles and posting dates is also at acceptable levels. The lower performance for dates can be attributed to various layouts used by blogs: dates can appear along with the title, below the title, along with the author name, or at the end of a post, constituting their accurate detection difficult. In some cases, dates may not be present in a blog (these cases were counted as an extraction failure of BlogBuster during evaluation). Finally, depending on the blog, the date may contain additional information along with the blog post publishing date, i.e. the name of the author or the number of comments. This can happen if all this all extracted textual information was contained inside a single DOM node, which is the finest level of detail the proposed approach can achieve. These cases were evaluated as correct, as the date was contained in the extracted text.

The performance of BlogBuster on blogs hosted in Wordpress can be seen in the following table:

	Precision	Recall	F-Measure
Blog posts	100.00 %	96.43 %	98.18 %
Title	77.78 %	75.00 %	76.36 %
Date	29.63 %	28.57 %	29.09 %

Table 3: Performance on blogs from Wordpress.

The performance of extracting individual blog posts remains at adequate levels, with titles and especially dates exhibiting lower performance. This decrease in performance can be attributed to the fact that fewer blogs from Wordpress were examined during the rule construction phase, as only 10 blogs hosted in Wordpress were used.

The third evaluation experiment was conducted on the entire corpus available, as mentioned in Table 1 (including blogs from Blogger and Wordpress). The evaluation results can be seen in Table 4.

	Precision	Recall	F-Measure
Blog posts	100.00 %	52.64 %	68.98 %
Title	90.41 %	47.60 %	62.36 %
Date	31.51 %	16.59 %	21.73 %

Table 4: Performance on all blogs of the evaluation corpus (Table 1).

The corpus contains a large percent (~80 %) of blogs that are not hosted in the platforms used during rule construction, allowing the possibility to be hosted/generated by alternative CMSs. BlogBuster was able to extract individual blog posts and titles with high accuracy, achieving at the same time a recall around 50 %. Performance of extracting dates was lower, but this is expected given the positional variation that dates exhibit among blogs.

7. Conclusions – Future Work

In this paper we presented BlogBuster, a tool for extracting a corpus from the blogosphere. Several general purpose approaches for removing boilerplate have been presented in the literature; however the blogosphere poses

[6] http://www.technorati.com/

additional requirements, such as a finer control over the extracted textual segments in order to accurately identify important elements, i.e. individual blog posts, titles, posting dates or comments. Besides boilerplate elimination, BlogBuster attempts to identify and convert into text such information, following a rule-based approach. A small set of rules was manually constructed, guided by observations made from a limited set of blogs. The corpus used during the rule construction phase contained randomly selected blog pages from blogs hosted in the Blogger and Wordpress hosting platforms.

BlogBuster was evaluated on a corpus different than the one used for rule construction. The evaluation corpus was collected from blogs returned by the Google's and Technorati's search APIs. Queries have been formed from words contained in news articles, collected from online news agencies. Blogs that were hosted in either Blogger or Wordpress were randomly removed from the results, until the percentage of the remaining ones was about 20 % of the corpus. Evaluation results showed that BlogBuster is very accurate in extracting individual blog posts and their titles from the evaluation corpus, with a recall approaching 50 % for both tasks. Identifying and extracting dates was less accurate, exhibiting an F-measure around 20 %. In addition, BlogBuster exhibits increased stability over malformed HTML pages, as it is build on top of a popular browser, Mozilla Firefox. Incorporating such a complete and full-featured browser allows the processing of dynamically generated content, i.e. generated through Javascript.

As future work, we plan to automate the acquisition of rules used by the extraction engine from a small annotated corpus. We think that automatic acquisition may help in cases where more complex rules are required, such as the extraction of dates where manually constructed rules exhibit limited coverage. Also, we plan to distribute the evaluation corpus, once a suitable annotation scheme is designed, and proper obfuscation is applied on the actual content to cater for copyright issues.

The BlogBuster tool can be freely accessed as a web service for research purposes. Finally, a live demo of BlogBuster can be found at:
http://www.intellitech.gr/index.php/lang-en/solutions-mainmenu-28/blog-processing.

8. References

Baroni, M., Chantree, F., Kilgarri, A., Sharo, S. (2008). Cleaneval: a competition for cleaning web pages. In *Proceedings of the 4th Web as Corpus Workshop (WAC4)*, - Can we beat Google?. N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odjik, S. Piperidis, and D. Tapias, editors, *Proceedings of the 6th International Language Resources and Evaluation (LREC 2008)*. Marrakech, Morocco, 2008.

Chakrabarti, D., Kumar, R., and Punera, K. (2007). Page-level template detection via isotonic smoothing. In *Proceedings of the 16th International Conference on World Wide Web*, pp. 61 – 70, New York, USA, 2007.

Clark, A., Guillemot, M. (2002). The CyberNeko HTML Parser. http://nekohtml.sourceforge.net/

CLEANEVAL (2007). http://cleaneval.sigwac.org.uk/

Evert, S. (2008). A lightweight and efficient tool for cleaning web pages. In *Proceedings of the 6th International Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco, May, 2008. http://www.lrec-conf.org/proceedings/lrec2008/.

Ferraresi, A., Zanchetta, E., Baroni, M., and Bernardini, S. (2008). Introducing and evaluating UKWAC, a very large web-derived corpus of English. In *Proceedings of the 6th International Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco, May, 2008. http://www.lrec-conf.org/proceedings/lrec2008/.

Kohlschütter, C., Fankhauser, P., and Nejdl, W. (2010). Boilerplate Detection using Shallow Text Features. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, New York City, NY USA, 2010.

Petasis, G., Fragkou, P., Theodorakos, A., Karkaletsis, V., Spyropoulos, C. D (2008). Segmenting HTML pages using visual and semantic information. In *Proceedings of the 4th Web as a Corpus Workshop (WAC2008), 6th International Language Resources and Evaluation Conference (LREC 2008)*, pp. 18 – 25, Marrakech (Morocco), June 1, 2008.

Raggett, D. (2000). The HTML Tidy Library Project. http://tidy.sourceforge.net/, http://www.w3.org/People/Raggett/tidy/

Spinn3r, (2007). Indexing the blogosphere. http://spinn3r.com/

Spousta Miroslav, Marek Michal, and Pecina Pavel (2008). Victor: the Web-Page Cleaning Tool. In *Proceedings of the 4th Web as Corpus Workshop (WAC4)*, - Can we beat Google?. *6th International Language Resources and Evaluation Conference (LREC 2008)*. Marrakech, Morocco, 2008.

W3C, (2000). XHTML™ 1.0: The Extensible HyperText Markup Language. http://www.w3.org/TR/xhtml1/

WAC4 (2008). The 4th Web as Corpus Workshop, *6th International Language Resources and Evaluation Conference (LREC 2008)*, Marrakech, Morocco, 1 June 2008. http://webascorpus.sourceforge.net/PHITE.php?sitesig=CONF&page=CONF_40_WAC-4___lb__2008__rb__

WAC5 (2009). The 5th Web as Corpus Workshop, San Sebastian, Basque Country, Spain, 7 September 2009. http://www.sigwac.org.uk/wiki/WAC5

Yi, L., Liu, B., and Li, X., (2003). Eliminating noisy information in web pages for data mining. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 296 – 305, 2003.